The Supervisory Control Problem of Discrete Event Systems using Polynomial Methods

Hervé Marchand, Michel Le Borgne

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Supervisory Control Problem of Discrete Event Systems using Polynomial Methods

Hervé Marchand, Michel Le Borgne

N°3790

Octobre 1999

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--3790--FR+ENG

INRIA
RENNES

The Supervisory Control Problem of Discrete Event Systems using Polynomial Methods

Hervé Marchand, Michel Le Borgne

Thème 1 — Réseaux et systèmes
Projet EP-ATR

Rapport de recherche n ° 3790 — Octobre 1999 — 58 pages

Abstract: This paper regroups various studies achieved around polynomial dynamical system theory. It presents the basic algebraic tools for the study of this particular class of discrete event systems. The polynomial dynamical systems are defined by polynomial equations over $\mathbb{Z}/3\mathbb{Z}$. Their study relies on concept borrowed from elementary algebraic geometry: varieties, ideals and morphisms. They are the basic tools that allow us to translate properties or specifications from a geometric description to suitable polynomial computations. In this paper, we more precisely describe the controller synthesis methodology. We specify the main requirements as simple properties, named control objectives, that the controlled plant has to satisfy.The plant is specified as a polynomial dynamical system over $\mathbb{Z}/3\mathbb{Z}$. The control of the plant is performed by restricting the controllable input values to values suitable with respect to the control objectives. This restriction is obtained by incorporating new algebraic equations into the initial polynomial dynamical system, which specifies the plant. Various kind of control objectives are considered, such as ensuring the *invariance* or the *reachability* of a given set of states, as well as partial order relation to be checked by the controlled plant.

Key-words: Discrete Event Systems, Polynomial Dynamical Systems,
Supervisory Control Problem, Optimal Control, Polynomial Methods.

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Le problème de la synthèse de contrôleurs sur des systèmes à événements discrets par des méthodes polynomiales

Résumé : Ce papier regroupe diverses études réalisées autour des systèmes dynamiques polynomiaux. Il présente les différents outils utilisés pour l'étude de cette classe particulière de systèmes à événements discrets. Ceux-ci sont définis par une famille d'équations polynomiales dans $\mathbb{Z}/3\mathbb{Z}$. Les concepts utilisés sont empruntés à la géométrie algébrique élémentaire: variétés, idéaux, morphismes. Dans ce papier, nous regardons plus précisément le problème de la synthèse de contrôleur. Dans un premier temps, le système est spécifié par un système dynamique polynomial. Le contrôle de ce système s'effectue alors en imposant des contraintes supplémentaires sur les entrées de celui-ci. Différents types d'objectifs de contrôle peuvent être considérés : assurer l'invariance, l'atteignabilité ou l'attractivité d'un ensemble d'états. Nous présentons également une théorie de la commande optimale, permettant de synthétiser des contrôleurs traduisant un critère qualitatif et non plus logique. Les objectifs de commande s'expriment alors comme des relations d'ordre ou comme un critère de minimisation sur une trajectoire bornée du système.

Mots-clé : Systèmes à événements discrets, systèmes dynamiques polynomiaux, problème de la synthèse de contrôleurs, contrôle optimal, méthodes polynomiales

Introduction & Motivations

Polynomial dynamical systems over $\mathbb{Z}/3\mathbb{Z}$ [1] were first introduced for studying the logical and synchronization of SIGNAL programs. The SIGNAL language is dedicated to reliable specifications of real-time reactive systems [9, 43]. SIGNAL describes systems (called processes) that communicate with their environment through a finite set of input and output ports. These processes are characterized by the sequences of communication events one can observe on the ports. For each event, some ports may receive or emit a value while some others do not participate in communication (*i.e.*, no value is present on these ports, for this particular event). The logical and synchronization part of a SIGNAL program is translated into a polynomial dynamical system over $\mathbb{Z}/3\mathbb{Z}$ on which various studies can be performed [42]. Polynomial functions over $\mathbb{Z}/3\mathbb{Z}$ provides us with efficient algorithms to represent these functions and polynomial equations. Hence, instead of enumerating the elements of sets and manipulating them explicitly, this approach manipulates the polynomial functions characterizing their set. This way, various properties can be efficiently proved on polynomial dynamical systems. It rapidly appeared that these aspects of SIGNAL programs were closely related to a certain class of Discrete Event Systems (DES).

In the Ramadge and Wonham theory of DESs [63, 64, 65], the behavior of a DES is modeled by the set of all its possible executions, i.e. by infinite sequences of events $a_1, a_2, \ldots$, where each a_i is an element of the finite alphabet that represents the possible actions of the system. At this point, the system is represented by some automaton, a finite state automaton for example. Given a plant (P) and a specification of the waited behavior (S), the control of the plant is performed by inhibiting some events belonging to a set of controlled events while the other events cannot be prevented from occurring (they are said to be uncontrollable) in such a way that the behavior of the controlled plant is included (in the sense of language) in the one of (S). It appears that this point of view although general is not always the most convenient from a practical point of view. Indeed in some situations, the problem is rather different and the relations between the plant and the controller are better described by considering the plant as emitting signals to the controller which in turn emits some other signals to control the plant. These signals are controllable and given as inputs for the plant [6, 40, 26].

As SIGNAL is a data-flow language, it is natural for us to use the input/output approach (however systems defined as finite state automata, like in [63], can also be considered within this framework). Even if it is possible to mix the two approaches in a single model, we feel that the two approaches are complementary in applications. So, polynomial dynamical systems offer an alternative (although closely related) approach to the classical automata theory. The plant is then represented by a polynomial dynamical system while the control of the plant

[1] $\mathbb{Z}/3\mathbb{Z}$ denotes the Galois field with 3 elements $\{-1, 0, +1\}$ with the usual multiplication and addition modulo 3 ($1 + 1 = -1$, and $3 = 0$).

is performed by restricting the controllable input values to values suitable for the control objective. This restriction is obtained by incorporating new algebraic equations to the plant modeled by a PDS in order to obtain the controlled plant. Various kind of control objectives can be solved within this framework.

- Historically, for a practical point of view, control objectives were expressed in terms of *invariance*, *reachability* and *attractivity* of a given set of states.
- However some control objectives cannot be expressed as traditional objectives, because they consider the way to reach a given logical goal. One way to solve these problems is to express them as partial order relations over the states of the plant.

Moreover, the traditional theory provides algorithms allowing the automatic synthesis of controllers according to the plant and their specification. However, the implementation of these algorithms is most of the time explicit. It renders the computation of the supervisory controllers not practical because of the size of the generated state spaces which is often too important when dealing with realistic applications (even if the proposed algorithms are often polynomial in the number of states). *A contrario*, the present approach proposes "symbolic" computations, based on a representation of the plant,by means of polynomials and at a lower level by Ternary decision diagrams (TDD) a slight extension of the Binary decision diagrams [15]. Hence, what we proposed here is an environment dedicated to 1) the specification of real-time plant 2) the automatic synthesis of supervisory controllers 3) the simulation of the result and 4) the automatic generation of executable code. Some others similar works can be found in [31, 6, 59, 3].

The present paper is organized as follows. In Section 1, we introduce the framework of polynomial dynamical systems, *i.e.*, the model, the mathematical framework and an overview of verification techniques. Section 2 is devoted to the Supervisory Control Problem presentation. In Section 3, solutions to various control problems are also given. We present traditional control objectives (e.g. ensuring the invariance of a set of states) and control objectives expressed as a partial order relation over states. In Section 4, the case of dynamical controller synthesis problem is explained. in Section 5, we present the integration of these techniques in the SIGNAL environment. Finally in Section 6, these methods are applied to the incremental design of a power transformer station controller. We conclude this paper by an appendix, in which we give a sketch of the actual implementation of the basic algebraic operations.

1 Polynomial dynamical systems

The first point concerns the choice of the model used to specify the plant. We have chosen to represent it by a polynomial dynamical system (PDS), which can be seen as an intentional representation of an automaton. Such a PDS originates from the logical abstraction of a

SIGNAL program (See Section 5.2.1). This abstraction is *automatically* performed by the SIGNAL compiler [42]. The resulting PDS can then be used as a formal basis for verification and optimal controller synthesis purposes. Using a high level language such as SIGNAL allows us to easily specify the real time system on which control has to be performed.

1.1 General form of PDS

A Polynomial Dynamical System (PDS) over Galois fields [39, 40, 41] (in our case $\mathbb{Z}/3\mathbb{Z}$ [2]) may be specified as a set of polynomial equations of the form:

$$S = \begin{cases} r_1(X,Y,X') = 0 \\ \dots \\ r_n(X,Y,X') = 0 \end{cases} \quad (1)$$

where X, Y, X' are vectors of variables in $\mathbb{Z}/3\mathbb{Z}$ and $dim(X) = dim(X') = n$.
Such a system will be denoted as:

$$R(X,Y,X') = 0 \quad (2)$$

where X, X' are vectors of variables in $\mathbb{Z}/3\mathbb{Z}$ and $dim(X) = dim(X') = n$. The vectors X and X' respectively encode the current and next states of the system and are called *state variables* (the states of the system contain the memory necessary for describing the system behavior), whereas Y is a vector of m variables in $\mathbb{Z}/3\mathbb{Z}$, called *event variables*. Such a dynamical system is generally implicit, or equivalently, behavioral in the sense of J.C. Willems [71], and no distinction between inputs and outputs is made. A case of particular interest is when it is possible to express the previous polynomial dynamical system (2) as follows:

$$S = \begin{cases} X' = P(X,Y) \\ Q(X,Y) = 0 \\ Q_0(X) = 0 \end{cases} \quad (3)$$

Such a polynomial dynamical system is called explicit[3]. In the sequel, we focus our attention on explicit polynomial dynamical system.

- The first equation of (3), $X' = P(X,Y)$, is called the *state transition equation*; it can be considered as a vector-valued function $[P_1, \dots, P_n]$ from $(\mathbb{Z}/3\mathbb{Z})^{n+m}$ to $(\mathbb{Z}/3\mathbb{Z})^n$. Each polynomial component P_i represents the evolution of the state variable X_i. It characterizes the dynamic of the system.

[2] The theory is the same for each finite field $\mathbb{Z}/p\mathbb{Z}$, with p prime

[3] Note that we can reformulate the explicit form (3) into two equations which are then combined by an $\wedge$ operator ($\oplus$ in our framework, see relation (7)): $R(X,Y,X') = (X' - P(X,Y)) \oplus Q(X,Y)$. The initial conditions are not taken into account in this case.

- The second equation of (3), $Q(X,Y) = 0$, is called the *constraint equations*: it is a system of equations $[Q_1, \ldots, Q_l]$, characterizing the *static* part of the system. It comprises all the equations characterizing the properties of the system involving the current instant only and codes the *static* part of the system (invariant for all instants t).
- The third equation, $Q_0(X) = 0$, is called the *initialization equation*; it is also a system of equations $[Q_{0_1}, \ldots, Q_{0_n}]$ which defines the set of initial states. If the *initialization equation* is not given, then all the states of the system are initial.

In the sequel, $x, x_t \in (\mathbb{Z}/_{3\mathbb{Z}})^n$, $y, y_t \in (\mathbb{Z}/_{3\mathbb{Z}})^n$ will denote particular instantiations of the vectors X and Y. Each $(x,y) \in (\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ s.t. $Q(x,y) = 0$ is said to be *admissible* and an event y is admissible in the state x if (x,y) is admissible. The sequence generated by a PDS, that is, all the sequences $(x_i, y_i)_{i\in\mathbb{N}}$ over $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ such that $Q_0(x_0) = 0$ and for all $i \in \mathbb{N}$, (x_i, y_i) is admissible and $x_{i+1} = P(x_i, y_i)$, are called trajectories of the system.

That way, a PDS can be seen as a finite state transition system[4]. The initial states of this transition system are the solutions of the equation $Q_0(X) = 0$. When the system is in a state $x \in (\mathbb{Z}/_{3\mathbb{Z}})^n$, any event $y \in (\mathbb{Z}/_{3\mathbb{Z}})^m$ such that $Q(x,y) = 0$ can trigger a transition leading to state $x' = P(x,y)$ (noted $x \xrightarrow{y} x'$).

Example 1 *We give here an example of polynomial dynamical system over* $\mathbb{Z}/_{3\mathbb{Z}}$. *This system is composed by 2 state variables, namely* X_1 *and* X_2 *and an event variable* Y. *The constraint equation is given by 4 polynomials*

$$\left\{\begin{array}{lcl} Q_1(X_1,X_2,Y) & = & (-X_1^2 + X_1)Y + X_1^2 - X_1, \\ Q_2(X_1,X_2,Y) & = & (X_2^2 - X_2)Y - X_2^2 + X_2, \\ Q_3(X_1,X_2,Y) & = & 1 - Y^2, \\ Q_4(X_1,X_2,Y) & = & (X_1 - 1)X_2^2 + (-X_1^2 + 1)X_2 + X_1^2 - X_1. \end{array}\right. \tag{4}$$

The state transition system is the following:

$$\left\{\begin{array}{lcl} X_1' = P_1(X_1,X_2,Y) & = & X_1X_2Y + X_2Y - Y - X_2^2 - X_1^2X_2 - 1 \\ X_2' = P_2(X_1,X_2,Y) & = & -X_1X_2Y + X_2Y + X_1Y + Y + X_2^2 - 1 \end{array}\right. \tag{5}$$

This dynamical system may be pictured as a finite automaton as in the figure (1). All points of $(\mathbb{Z}/_{3\mathbb{Z}})^2$ *but* $(0,-1)$ *and* $(-1,0)$ *are admissible states.*

The labeled arrows picture the transitions and the corresponding events. For example, in state $(0,1)$ *the events* 1 *and* -1 *are admissible since* $(0,1,-1)$ *and* $(0,1,1)$ *are solutions of*

[4] A finite state transition system is a quadruple $(E, A, I, \rightarrow)$ where E and A are finite sets, I is a subset of E, and $\rightarrow$ is a ternary relation on $E \times A \times E$. E is the set of states, A the set of events, I the set of initial states and $\rightarrow$ is the transition relation. Starting from a state in I, such a system evolves by executing transitions: when the system is in a state x, it can move to a state x' if there exists an event y such that (x, y, x') are in relation by $\rightarrow$ (noted $x \xrightarrow{y} x'$) [2].

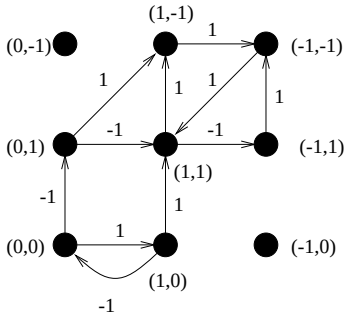


Figure 1: The dynamical system S.

equations (4); the arrow with origin $(0,1)$ *and label* 1 *indicates that* $P((0,1),1) = (1,-1)$. *Similarly* $P((0,1),-1) = (1,1)$. □

Finally, like automata, PDSs can be composed in the following way: let us consider two PDSs S_1 and S_2, then the composition $S_1 \| S_2$ (assuming that the two systems have no common state variable) is simply obtained by combining the equations together. The event variables with the same label in the two PDSs are considered as identical as is usual in mathematics.

1.2 The Algebraic Framework

The theory of PDSs uses concepts of algebraic geometry such as *ideals*, *varieties* and *morphisms*[5]. In this theory, we can link properties of algebraic sets with properties of the ideals, i.e. sets of polynomials [40, 38]. Similar works may be found in [25, 26, 19]. Finally, an overview of ideals and varieties is described in [21].

1.2.1 Preliminary notions

We wish to work with polynomials in several variables, with coefficients in $\mathbb{Z}/3\mathbb{Z}$. The notation Z will denote a collection of formal variables $Z_1, \ldots Z_l$, and similarly for X, Y, etc (note that in the sequel X (resp. Y) will generally denote the set of state variables (resp. event variables)). The ring[6] of the polynomials in the variables $Z = (Z_1, \ldots Z_l)$ with coefficients in $\mathbb{Z}/3\mathbb{Z}$ will be denoted by $\mathbb{Z}/3\mathbb{Z}[Z]$.

However we need that for each variable Z_i the condition $Z_i^3 = Z_i$ be satisfied, expressing that we indeed consider Z_i as a variable living in $\mathbb{Z}/3\mathbb{Z}$. Indeed, due to Fermat's Theorem, the polynomial function $Z_i^3 - Z_i$ is identically null over $\mathbb{Z}/3\mathbb{Z}$. Then, we naturally introduce

[5] Good textbooks on these subjects are [4, 70, 58, 60, 66]

[6] A commutative ring R is a set of elements and two operations, + and *, both are commutative, associative, distributive and closed in R. + and * have identity elements, 0 and 1 respectively. For every element $a \in R$ there exists an element $b \in R$ such that $a + b = 0$, *i.e.* + has an inverse.

the quotient ring of polynomial functions $A[Z] = \mathbb{Z}/_{3\mathbb{Z}}[Z]/_{<Z^3-Z>}$, where all polynomials $Z_i^3 - Z_i$ are identified to zero, written $Z^3 - Z = 0$. Equivalently, $A[Z]$ can be regarded as the set of polynomial functions with coefficients in $\mathbb{Z}/_{3\mathbb{Z}}$ for which the degree in each variable is lower than 2. For example, let $P(X_1, Y_1) = X_1 + 4Y_1 + 2X_1 * Y_1 + X_1^4 Y_1 - 4Y_1^3$ be a polynomial function in the ring $\mathbb{Z}/_{3\mathbb{Z}}[X_1, Y_1]$. Then the corresponding polynomial function in the quotient ring $A[X_1, Y_1]$ is given by $P(X_1, Y_1) = X_1 - X_1 * Y_1 + X_1^2 Y_1$.

1.2.2 Ideals and Varieties

Let E be a set of elements in $(\mathbb{Z}/_{3\mathbb{Z}})^l$. The set $\mathcal{I}(E)$ of polynomials defined by:

$$\mathcal{I}(E) \stackrel{def}{=} \{p \in A[Z] \ / \ \forall z \in E,\ p(z) = 0\} \tag{6}$$

is called the *ideal* of the set E in $A[Z]$. This set represents all the polynomials, for which the set E is a solution. More generally, an ideal $\mathcal{I} =< g_1, \ldots, g_l > \subseteq A[Z]$ is a set of polynomials, where each $f \in \mathcal{I}$ can be written as

$$f(Z) = \sum_{i=1}^{l} h_i(Z) * g_i(Z),$$

where h_i are polynomials of $A[Z]$. The polynomials $g_1, \ldots, g_l$ are called the generators of the ideal $\mathcal{I}$[7]. In fact, it is possible to represent an ideal $\underline{a}$ by a single polynomial, called *principal generator*:

Proposition 1 *let $\underline{a} =< g_1, \ldots, g_k >$ be an ideal of $A[Z]$. Then, the polynomial function $f = g_1 \oplus g_1 \oplus \ldots \oplus g_k$ is a principal generator of $\underline{a}$ meaning that $\underline{a} =< f >$, where the operator $\oplus$ is defined by*

$$f \oplus f' \stackrel{def}{=} (f^2 + f'^2)^2. \tag{7}$$

◇

A sketch of ideal implementation is given in Appendix A.

Remark 1 *The sum of $\underline{a}_1 =< g_1 >$ and $\underline{a}_2 =< g_2 >$ is an ideal $\underline{a}_{12} = \underline{a}_1 + \underline{a}_2 =< g_1, g_2 >$.* ∘

Conversely, to any set of polynomials $G \in A[Z]$, we can associate a set in $(\mathbb{Z}/_{3\mathbb{Z}})^l$, called the *variety* of G, defined by:

$$\mathcal{V}(G) \stackrel{def}{=} \{z \in (\mathbb{Z}/_{3\mathbb{Z}})^l \ / \ \forall p \in G,\ p(z) = 0\} \tag{8}$$

[7] an ideal $\mathcal{I}$ can also be considered as a subset of $A[Z]$ satisfying (1) $\mathcal{I}$ is as ring in itself, *i.e.* closed under arbitrary addition and multiplication (2) if $p \in \mathcal{I}$ then $qp \in \mathcal{I}$ for all $q \in A[Z]$.

This set represents all the common zeros for a given set of polynomials.

The advantage of using ideals is that there is a direct correspondence between an ideal and the associated variety. In fact, due to the finiteness of the field $\mathbb{Z}/_{3\mathbb{Z}}$, these relations are very simple. Hence the following fundamental theorem:

Theorem 1 *In $A[Z]$ where Z is a collection of "l" variables, for every ideal $\underline{a}$ of $A[Z]$, and every variety $V \in (\mathbb{Z}/_{3\mathbb{Z}})^l$, we have:*

$$\mathcal{V}(\mathcal{I}(V)) = V \text{ and } \mathcal{I}(\mathcal{V}(\underline{a})) = \underline{a} \tag{9}$$

◇

The proof of this theorem can be found in [38]. The first relation of (9) means that any subset of $(\mathbb{Z}/_{3\mathbb{Z}})^l$ is a variety and that $\mathcal{I}(V)$ is characteristic of V (*i.e.*, if $V_1 \neq V_2$ then $\mathcal{I}(V_1) \neq \mathcal{I}(V_2)$). The second one allows a set of generators to be obtained from equations defining $E = \mathcal{V}(\underline{a})$. The study of PDS strongly relies on Theorem 1. Indeed, properties of states or of events are translated to equivalent properties of their characteristic ideals. The next proposition shows how relations and operations over varieties can be translated into relations/operations between associated ideals.

Proposition 2 *[66] Let V_1 and V_2 be two varieties of $(\mathbb{Z}/_{3\mathbb{Z}})^l$, then:*

1. $V_1 \subseteq V_2 \Leftrightarrow \mathcal{I}(V_1) \supseteq \mathcal{I}(V_2)$
2. $\mathcal{I}(V_1 \bigcap V_2) = \mathcal{I}(V_1) + \mathcal{I}(V_2)$
3. $\mathcal{I}(V_1 \bigcup V_2) = \mathcal{I}(V_1) \bigcap \mathcal{I}(V_2)$ ◇

Hence, instead of enumerating sets and manipulating them explicitly, this approach manipulates the polynomial functions characterizing their set. Finally, we can remark that for any subset E of $(\mathbb{Z}/_{3\mathbb{Z}})^n$, $\mathcal{I}(E)$ is an ideal of $A[X]$ and the characteristic ideal of $E \times (\mathbb{Z}/_{3\mathbb{Z}})^m$ is the ideal generated by $\mathcal{I}(E)$ in $A[X,Y]$ written $\mathcal{I}(E)A[X,Y]$ in the sequel.

1.2.3 Elimination of variables

The projection is another useful operation on varieties. Consider a subset E of state/event pairs (x,y) in $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$. We write $\exists elim_Y(E)$ for the projection of E onto the components corresponding to the variables X. This is defined by:

$$\exists elim_Y(E) \stackrel{def}{=} \{x / \exists y \in (\mathbb{Z}/_{3\mathbb{Z}})^m, (x,y) \in E\} \tag{10}$$

Proposition 3 $\mathcal{I}(\exists elim_Y(E)) = \mathcal{I}(E) \cap A[X]$.

Proof: The inclusion $\mathcal{I}(E) \cap A[X] \subseteq \mathcal{I}(\exists elim_Y(E))$ is obvious. Conversely, consider a polynomial $p(X) \in A[X]$ such that $\forall x \in \exists elim_Y(E), p(x) = 0$ and $(x, y) \in E$. So x belongs to the projection of E and then $p(x) = 0$. But $p(x)$ can also be considered as a polynomial in X and Y that is then null on E. ◇

Now, given a set of state E and g_E, its principal generator, we will note $\exists elim_Y(< g_E >)$ (or simply $\exists elim_Y(g_E)$), the operation which performs the projection on the ideal associated with the variety E.

Notice that $A[X, Y] \cong (A[X])[Y]$, hence a polynomial $p \in A[X, Y]$ may be rewritten as a polynomial in the variables Y with polynomials in X as coefficients.

Definition 1 *Let $p \in A[X, Y]$, then $\forall elim_Y(p)$ is the ideal of $A[X]$ spanned by the coefficients of p when rewritten as a polynomial in the variables Y. $\forall elim_Y(\underline{a})$ is defined accordingly for $\underline{a} \subseteq A[X, Y]$.* •

Given a principal generator $g(X, Y)$ of $\underline{a}$, it is easy to prove that the ideal $\forall elim_Y(\underline{a})$ is generated by $\forall elim_Y(g)$. Its geometric meaning is captured by the relation:

Proposition 4 $\mathcal{V}(\forall elim_Y(\underline{a})) = \{x : \forall y, (x, y) \in \mathcal{V}(\underline{a})\}$.

Proof: The inclusion $\subseteq$ is obvious. Reciprocally, if x is such that $(x, y) \in \mathcal{V}(\underline{a})$ for all y, let $P(X, Y) = \sum \alpha(X)\beta(Y)$ a polynomial function of $\underline{a}$. Then, the polynomial function $P(x, Y)$ is identically null over $(\mathbb{Z}/_{3\mathbb{Z}})^m$ and all the coefficients $\alpha(x)$ are then null. ◇

1.2.4 Operations on dynamical behaviors.

To capture the dynamical aspect of a PDS, we introduce the notion of comorphism. The equation $X' = P(X, Y)$ of (3) is of the form $\{X'_i = P_i(X, Y)\}_{i \in [1..n]}$, where n is the number of state variables and $P_1, \ldots, P_n$ are polynomials in $A[X, Y]$. P can be considered as a polynomial function from $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ to $(\mathbb{Z}/_{3\mathbb{Z}})^n$. With P, there is an associated *comorphism* P^* from $\mathbb{Z}/_{3\mathbb{Z}}[X]$ to $\mathbb{Z}/_{3\mathbb{Z}}[X, Y]$, defined by: for a polynomial $q \in \mathbb{Z}/_{3\mathbb{Z}}[X]$:

$$\begin{aligned} P^*(q(X)) &= P^*(q(X_1, \ldots, X_n)) \\ &\stackrel{def}{=} q(P_1(X, Y), \ldots, P_n(X, Y)) \end{aligned} \tag{11}$$

In other words, $P^*(q(X))$ is obtained by substituting every X_i in q with the corresponding $P_i(X, Y)$. There are relations between varieties and ideals using morphisms and comorphisms that are used to perform computations on PDSs.

Proposition 5 *If E is a subset of $(\mathbb{Z}/_{3\mathbb{Z}})^n \times (\mathbb{Z}/_{3\mathbb{Z}})^m$ and $\underline{b}$ an ideal of $A[X]$, then*

$$\mathcal{I}(P(E)) = P^{*-1}(\mathcal{I}(E)), \tag{12}$$

$$\mathcal{V}(< P^*(\underline{b}) >) = P^{-1}(\mathcal{V}(\underline{b})). \tag{13}$$

Proof:

- Let $q \in P^{*-1}(\mathcal{I}(E))$ then $P^*(q) \in \mathcal{I}(E)$. This is equivalent to saying that $\forall (x,y) \in E,\ q(P(x,y)) = 0$, which in turns implies $q \in \mathcal{I}(P(E))$, and finally $P^{*-1}(\mathcal{I}(E)) \subseteq \mathcal{I}(P(E))$.

  Reciprocally, let $q \in \mathcal{I}(P(E))$. This means that $\forall (x,y) \in E,\ q(P(x,y)) = 0 = P^*(q(x,y))$. Then, $P^*(q) \in \mathcal{I}(E)$. It implies $q \in P^{*-1}(P^*(q)) \subseteq P^{*-1}(\mathcal{I}(E))$, and finally $\mathcal{I}(P(E)) \subseteq P^{*-1}(\mathcal{I}(E))$.

- Let $(x,y) \in P^{-1}(\mathcal{V}(\underline{b}))$, then $P(x,y) \in \mathcal{V}(\underline{b})$. So, $\exists q \in \underline{b},\ q(P(x,y)) = P^*(q(x,y)) = 0$. In particular, this says that $(x,y) \in \mathcal{V}(P^*(\underline{b}))$ and that $P^{-1}(\mathcal{V}(\underline{b})) \subseteq \mathcal{V}(< P^*(\underline{b}) >)$.

  Reciprocally, assume that the couple (x,y) belongs to $\mathcal{V}(P^*(\underline{b}))$, then $\forall q \in \underline{b},\ P^*(q(x,y)) = 0$. Consider now $x = P(x,y)$, then $\forall q \in \underline{b},\ q(x) = 0$. This means that $x \in \mathcal{V}(\underline{b})$ and then $(x,y) \in P^{-1}(x) \subseteq P^{-1}(\mathcal{V}(\underline{b}))$. Finally, $\mathcal{V}(< P^*(\underline{b}) >) \subseteq P^{-1}(\mathcal{V}(\underline{b}))$. ◇

Moreover, in the quotient ring $A[X,Y]$, using relations (13) and (9) we can easily prove that

Proposition 6 $\mathcal{I}(P^{-1}(E)) = < P^*(g) >$, *where* $\mathcal{I}(E) =< g >$. ◇

This section gave an overview of the basic operators (see also [38, 40]) that will be used to perform verification of PDS properties as well as the automatic synthesis of controllers.

1.3 Basic Properties of PDS

Using the operations defined in the previous section, various kind of properties can be proved on PDSs. Most of them will be used in the sequel as control objectives for controller synthesis purposes. However, the validity of the controller synthesis approach heavily depends on the model of the plant. Therefore, we feel that verification techniques are still necessary to prove the correctness of the plant to be controlled.

1.3.1 Liveness

We say that a system is *live* if it cannot be in a state from which no transition can be taken. Roughly speaking, a system is live if "something can always happen". This property states that every trajectory of the system is infinite. In the PDS framework, the definition is formalized as follows.

Definition 2 *A state x is* live *if there exists an event y such that $Q(x,y) = 0$ (i.e., a transition can be taken). A set of states V is* live *if every element in V is live. Finally, a PDS is* live *if for all (x,y) such that $Q(x,y) = 0$, $P(x,y)$ is a live state.* •

The translation of Definition 2 in terms of ideals and varieties leads to the following theorem:

Theorem 2 *A PDS is live if and only if* $P^*(< Q > \cap A[X]) \subseteq < Q >$.

Proof: Let E be the set

$$E = \{x \in (\mathbb{Z}/_{3\mathbb{Z}})^n \ / \ \exists y \in (\mathbb{Z}/_{3\mathbb{Z}})^m,\ Q(x,y) = 0\} = \exists elim_Y(\mathcal{V}(< Q(X,Y) >)).$$

Following Definition 2, a system is live iff for all pairs (x, y) s.t. $Q(x, y) = 0$, the state $P(x, y)$ is in E. In other words,

$$\begin{array}{rrcl} & P(\mathcal{V}(< Q >)) & \subseteq & E = \exists elim_Y(\mathcal{V}(< Q >)) \\ \Leftrightarrow & \mathcal{I}(\exists elim_Y(\mathcal{V}(< Q >))) & \subseteq & \mathcal{I}(P(\mathcal{V}(< Q >))) \\ \Leftrightarrow & < Q > \cap A[X] & \subseteq & P^{*-1}(< Q >) \\ \Leftrightarrow & P^*(< Q > \cap A[X]) & \subseteq & < Q > \end{array}$$

The first equivalence translates set relations into ideal relations and reverses inclusion (Proposition 2, item 1). The second equivalence is derived from Proposition 3 and Relation (12) of Proposition 5. The fact that P^* is an increasing function justifies the last equivalence. ◇

1.3.2 The invariance of a set of states E

Informally, whereas liveness properties stipulate that some "good things" do happen, safety properties stipulate that " bad things" do not happen during any execution of the program [5]. In our framework, it is formalized using the notion of invariance:

Definition 3 *A set of states $E \in (\mathbb{Z}/_{3\mathbb{Z}})^n$ is* invariant *w.r.t. a PDS if and only if for every x in E and every y admissible in the state x, the state $x' = P(x, y)$ remains in E.* •

Definition 3 can be reformulated as follows:

Theorem 3 *A set E is invariant if and only if $P^*(\mathcal{I}(E)) \ \subseteq \ < Q > + \mathcal{I}(E)A[X,Y]$.*

Proof: By Definition 3, E is invariant iff $\forall x \in E, \forall y \in (\mathbb{Z}/_{3\mathbb{Z}})^m, Q(x,y) = 0 \Rightarrow P(x,y) \in E$, that can be rewritten

$$\begin{array}{rll} & E \times (\mathbb{Z}/_{3\mathbb{Z}})^m \cap \mathcal{V}(< Q >) \subseteq P^{-1}(E) & \\ \Leftrightarrow & P(\mathcal{V}(< Q >) \cap (E \times (\mathbb{Z}/_{3\mathbb{Z}})^m)) \subseteq E & \\ \Leftrightarrow & \mathcal{I}(E) \subseteq \mathcal{I}(P(\mathcal{V}(< Q > \cap (E \times (\mathbb{Z}/_{3\mathbb{Z}})^m)))) & (Proposition\ 2,\ item\ 1) \\ \Leftrightarrow & \mathcal{I}(E) \subseteq P^{*-1}(\mathcal{I}(\mathcal{V}(< Q >) \cap (E \times (\mathbb{Z}/_{3\mathbb{Z}})^m))) & (Proposition\ 5,\ relation\ (12)) \\ \Leftrightarrow & \mathcal{I}(E) \subseteq P^{*-1}(< Q > + \mathcal{I}(E)A[X,Y]) & (Proposition\ 2,\ item\ 2,\ Theorem\ 1) \\ \Leftrightarrow & P^*(\mathcal{I}(E)) \subseteq < Q > + \mathcal{I}(E)A[X,Y] & \end{array}$$

◇

This theorem states that the live states in E are included in the states that can reach E. Another characterization of the invariance can be obtained by introducing the operator $\widetilde{pre}$, defined by:

For any set of states F,

$$\widetilde{pre}\ (F) = \{x \in (\mathbb{Z}/_{3\mathbb{Z}})^n / \forall y \in (\mathbb{Z}/_{3\mathbb{Z}})^m, Q(x,y) = 0 \Rightarrow P(x,y) \in F\} \qquad (14)$$

Then, from Definition 3, it is clear that

Corollary 1 *A set of states E is invariant if and only if $E \subseteq \widetilde{pre}\,(E)$.* ◇

Remark 2 *Liveness, as defined earlier, is a particular case of invariance of the property of admissibility. The set $E = \mathcal{V}(< Q > \cap A[X])$ of live states of a PDS is invariant iff*

$$P^*(\mathcal{I}(E)) = P^*(\mathcal{I}(\mathcal{V}(< Q > \cap\, A[X]))) \subseteq < Q > + (< Q > \cap A[X])\ A[X,Y] =< Q > \quad (15)$$

In other words, a PDS is live if and only if the set of live states is invariant. ◦

The largest invariant subset of E: consider now some set of states E. As there exist some subsets of E that are invariant (at least one: the empty set) and as the union of two invariant subsets of states is again an invariant, there exists a largest subset of E that is invariant. The largest invariant subset included in E is the limit of the following decreasing sequence $(E_i)_{i\in\mathbb{N}}$ of finite sets:

$$\left\{ \begin{array}{rcl} E_0 & = & E \\ E_{i+1} & = & E_i \cap \widetilde{pre}\,(E) \end{array} \right. \quad (16)$$

Theorem 4 *Let E_{inv} be the result of the preceding fix-point computation, then E_{inv} is the largest invariant subset of E*

Proof: By construction $E_{inv} \subseteq E$ and $E_{inv} = E_{inv} \cap \widetilde{pre}\,(E_{inv})$, entailing $E_{inv} \subseteq \widetilde{pre}\,(E_{inv})$ and then E_{inv} is invariant. Now, we have to show that E_{inv} is the largest invariant of E. If $E_{inv} = E$ then E_{inv} is clearly the largest one. Otherwise, consider an invariant set $G \subseteq E$ with $G \not\subseteq E_{inv}$. Then, there exists a i with $G \subseteq E_i$ and $G \not\subseteq E_{i+1}$. As $\widetilde{pre}$ is a monotonic function, we have $\widetilde{pre}\,(G) \subseteq \widetilde{pre}\,(E_i)$. Because G is invariant, we have $G \subseteq \widetilde{pre}\,(G)$ and so $G \subseteq E_{i+1}$, which contradicts the hypothesis. ◇

In practice, we implement this computation over the ideals $(\underline{a}_i)_{i\in\mathbb{N}} = \mathcal{I}(E_i)_{i\in\mathbb{N}}$. First, we have

$$\mathcal{I}(\widetilde{pre}\,(E)) = \forall elim_Y(P^*(\mathcal{I}(E)) \cap < Q >^c)) \quad (17)$$

where $< Q >^c$ is the ideal associated with $(\mathbb{Z}/_{3\mathbb{Z}})^n - \mathcal{V}(< Q >)$. Finally, The computed sequence of ideals $(\underline{a}_i)_{i\in\mathbb{N}}$ is:

$$\left\{ \begin{array}{rcl} \underline{a}_0 & = & \mathcal{I}(E) \\ \underline{a}_k & = & \underline{a}_{k-1} \ + \ \forall elim_Y(P^*(\underline{a}_{k-1}) \cap < Q >^c) \end{array} \right. \quad (18)$$

Remark 3 *Note that the emptiness of the largest invariant subsets of a set of states E means that every trajectory entering E may eventually leave E. Thus, some fairness properties can be viewed as particular instances of invariance properties.*

1.3.3 Invariance under control

Another useful property is the one of *invariance under control* of a set of states E. Intuitively, E is invariant under control if we can modify the system, by adding new constraints, in such a way that E becomes invariant.

Definition 4 *E is invariant under control if for any state x in E, there exists an admissible event y s.t. $Q(x,y) = 0$ and $P(x,y) \in E$.* •

Whereas the invariance property of a set E states that every trajectory initialized in E never leaves E, the above definition means that it is possible starting from E to always stay in E.

Theorem 5 *E is invariant under control iff* $(< Q > + P^*(\mathcal{I}(E))) \cap A[X] \subseteq \mathcal{I}(E)$. ◇

The proof is similar to the one of Theorem 3. Roughly speaking, this theorem states that a set E is invariant under control if and only if E is included in the set of admissible states from which E can be reached.

Similarly to the largest invariant subset of a given set E, it is possible to compute the largest invariant under control subset of E. To do so, we replace $\widetilde{pre}$ by the pre operation,defined by

$$pre(F) = \{x \in (\mathbb{Z}/_{3\mathbb{Z}})^n / \exists y \in (\mathbb{Z}/_{3\mathbb{Z}})^m,\ Q(x,y) = 0 \wedge P(x,y) \in F\} \tag{19}$$

Non existing invariant under control in E means that any trajectory entering E must leave E after a finite delay. This can be used to check absence of *starvation*.

1.3.4 Derived properties

Many properties may be derived from these three basic properties.

Definition 5 *A set F is reachable for a PDS, if there exists a trajectory starting in the initial set of states E_0 that reaches F.* •

From Definition 5, we can derive the following proposition.

Proposition 7 *F is reachable for a PDS if the set of initial states is* not *included in the largest invariant subset of the* complement *of F.* ◇

Proof: Let E be the largest invariant subset of $\mathbb{Z}/_{3\mathbb{Z}}{}^n - F$, i.e. the complement of F in the state space, and let x_0 be an initial state. If $x_0 \in E$, then every trajectory starting from x_0 stays within E, so F cannot be reached from x_0. Conversely, if $x_0 \notin E$, then there is a trajectory starting from x_0 that eventually leaves E, i.e. reaches F. ◇

Definition 6 *Let E and F be two set of states. We say that F is attractive for E if every trajectory initialized in E reaches F.* •

Attractivity of a set of states F for a set of states E can be proved using following proposition:

Proposition 8 *F is attractive for E iff the set E is not contained in the largest invariant under control of the* complement *of F.* ◇

The proof is similar to the proof of Proposition 7.

This section presented an overview of the method for the verification of properties, with its basic operators. Moreover, using the algebraic methods, it is also possible to symbolically express CTL formulae [22], propositional μcalculus formulae [36, 62] as well as bisimulation equivalences [35]. For a more complete review of the theoretical foundation of this approach, the reader may refer to [38, 40]. Examples of verification can be found in [1, 42, 55].

2 Control of PDSs

There exist different theories for control of discrete event systems since the 80's [34, 65, 32, 33, 44, 45][8]. The starting point of these theories is based on the following: given a model for the system and the control objectives, a controller must be derived by various means such that the resulting behavior of the closed loop system meets the control objectives. In our framework, the system emits uncontrollable outputs, called signals, to the controller which in turn emits other signals to control the system (these signals are controllable and are considered as inputs for the system) [6, 40]. The specification of the system is represented by a PDS while the control of the system is described by incorporating new algebraic equations into the PDS. Similar works can be found in [26].

2.1 Controllable polynomial dynamical system

The control of a PDS relies on a distinction between events. we distinguish between the *uncontrollable* events which are sent by the system to the controller, and the *controllable* event which are sent by the controller to the system. Considering this distinction between controllable and uncontrollable events, a PDS is now written as follows:

$$S : \begin{cases} Q(X,Y,U) &= 0 \\ X' &= P(X,Y,U) \\ Q_0(X) &= 0 \end{cases} \tag{20}$$

where X represents the state variables; Y and U are respectively the sets of uncontrollable and controllable event variables, respectively. Such a PDS is called a *controllable polynomial dynamic system* (CPDS). Let n, m, and p be the respective dimensions of X, Y, and U.

[8] some other works dealing with controller synthesis for Timed Systems can be found in [49, 48, 3]

The trajectories of a CPDS are sequences $\{(x_t, y_t, u_t)\}$ in $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m+p}$ such that $Q_0(x_0) = 0$ and, for all t, $Q(x_t, y_t, u_t) = 0$ and $x_{t+1} = P(x_t, y_t, u_t)$. Each event (y_t, u_t) contains an uncontrollable component y_t and a controllable one u_t[9]. We have no direct influence on the y_t part which depends only on the state x_t, but we observe it. On the other hand, we have full control over u_t and we can choose any value of u_t which is admissible, *i.e.*, such that $Q(x_t, y_t, u_t) = 0$.

To distinguish the two components, a vector $y \in (\mathbb{Z}/_{3\mathbb{Z}})^m$ is called an *event* and a vector $u \in (\mathbb{Z}/_{3\mathbb{Z}})^p$ a *control*. From now on, an event y is *admissible* in a state x if there exists a control u such that $Q(x, y, u) = 0$; such a control is said *compatible* with y in x.

An important property of CPDS is the *U-invariance under control*. Its definition is adapted from Definition 4 to take into account the controllable and uncontrollable events.

Definition 7 *A set of states E is* U-invariant under control *for a CPDS S if, for every state $x \in E$ and every y admissible in x, there exists a compatible control u s.t. $P(x, y, u) \in E$.* •

This definition leads to the following proposition:

Proposition 9 *A set of states E is U-invariant under control if and only if*

$$(<Q> + <P^\star(\mathcal{I}(E))>) \cap A[X, Y] \subseteq \mathcal{I}(E)A[X, Y] + (<Q> \cap A[X, Y]). \tag{21}$$

Proof: First consider the set $\exists elim_U(\mathcal{V}(<Q>))$. Its characteristic ideal is $<Q> \cap \mathbb{Z}/_{3\mathbb{Z}}[X, Y]$. Let E be a U-invariant under control set of states, by definition, we have

$$\begin{aligned}
&\forall x \in E, y \in (\mathbb{Z}/_{3\mathbb{Z}})^m,\ (x, y) \in \exists elim_U(\mathcal{V}(<Q>)) \ \Rightarrow \\
&\qquad\qquad \exists u \in (\mathbb{Z}/_{3\mathbb{Z}})^p,\ Q(x, y, u) = 0 \text{ and } P(x, y, u) \in E \\
&\Leftrightarrow\ (E \times (\mathbb{Z}/_{3\mathbb{Z}})^m) \cap \exists elim_U(\mathcal{V}(<Q>)) \subseteq \exists elim_U(\mathcal{V}(<Q>) \cap P^{-1}(E)) \\
&\Leftrightarrow\ \mathcal{I}(\exists elim_U(\mathcal{V}(<Q>) \cap P^{-1}(E))) \subseteq \mathcal{I}((E \times (\mathbb{Z}/_{3\mathbb{Z}})^m) \cap \exists elim_U(\mathcal{V}(<Q>))) \\
&\Leftrightarrow\ (<Q> + <P^\star(\mathcal{I}(E))>) \cap A[X, Y] \subseteq \mathcal{I}(E)A[X, Y] + <Q> \cap A[X, Y]
\end{aligned}$$ ◇

Similarly to the invariance, another characterization of the U-invariance under control can be obtained by introducing the operator $\tilde{\tilde{pre}}$, defined for any set of states F by:

$$\tilde{\tilde{pre}}\,(F) = \{x \ / \ \forall y,\ \exists elim_U(Q(x, y, u)) = 0 \ \Rightarrow \exists u, Q(x, y, u) = 0 \text{ and } P(x, y, u) \in F\}. \tag{22}$$

Corollary 2 *A set E is U-invariant under control iff $E \subseteq \tilde{\tilde{pre}}\,(E)$* ◇

It is easy to see that the union of two U-invariant under control sets is also U-invariant under control. As a consequence, there exists a largest U-invariant under control subset of E. The computation of the largest U-invariant under control E is similar as the one presented in Section 1.3.2. To do so, it suffices to replace the $\tilde{pre}$ operator by $\tilde{\tilde{pre}}$. The actual computation of $\tilde{\tilde{pre}}\,(F)$ in terms of ideals is given in Appendix A.

[9] In our framework, the events are then partially controllable, whereas in [65], the events are either controllable or uncontrollable.

2.2 The controllers

The behavior of a CPDS is the following: at each instant t, given a state x_t and an admissible event y_t, we can choose some control u_t, such that $Q(x_t, y_t, u_t) = 0$. A CPDS S can then be controlled by first selecting a particular initial state x_0 and then by choosing suitable values for $u_1, u_2, \ldots, u_n, \ldots$.

Different strategies can be chosen to determine the value of the controls. We will here consider control policies where the value of the control u_t is statically computed from the value of x_t and y_t. Such a controller is called a *static controller*[10]. Formally, a static controller of a CPDS is of the form:

$$\left\{ \begin{array}{lcl} C(X,Y,U) & = & 0 \\ C_0(X) & = & 0 \end{array} \right. \tag{23}$$

where the equation $C_0(X) = 0$ determines initial states satisfying the control objectives and the other equation describes how to choose the static controls; when the controlled system is in state x, and when an event y occurs, any value u such that $Q(x, y, u) = 0$ and $C(x, y, u) = 0$ can be chosen; i.e. given a state x_t and an event y_t, choosing a u_t such that $C(x_t, y_t, z_t) = 0$ implies an evolution of the state in accordance with the control objective.

The behavior of the system S supervised by the controller is then modeled by the following system:

$$S_C : S \oplus (C, C_0) \;=\; \left\{ \begin{array}{l} X' = P(X,Y,U) \\ Q(X,Y,U) = 0 \\ C(X,Y,U) = 0 \\ Q_0(X) = 0 \\ C_0(X) = 0 \end{array} \right. \tag{24}$$

However, not every controller (C, C_0) is acceptable. First, the controlled system S_C has to be initialized ; thus, the equations $Q_0(X) = 0$ and $C_0(X) = 0$ must have common solutions. Furthermore, admissibility in S must imply admissibility in S_C, because of uncontrollability of events Y. Hence the following definition of an *acceptable controller*:

Definition 8 *An* acceptable controller *for a CPDS S is a controller as (23) which satisfies*

1. *The initial constraints $C_0(X) = 0$ and $Q_0(X) = 0$ have common solutions;*
2. *For each reachable x in S_C, any uncontrollable event y admissible in x for S is also admissible for the controlled system S_C.* •

[10] We present in Appendix 4 the case of dynamic controllers.

3 Control Objective Examples

We now illustrate the use of our techniques to solve particular classical control synthesis problems. In the historical development of control theory, control objectives are expressed in terms of *invariance*, *reachability* and *attractivity*. Then, control equations can be synthesized using the algebraic operations presented in Section 1.2 [23]. However, we also consider a new kind of control objectives based on partial orders over states that allows us to introduce optimality criterion for the control.

3.1 Traditional Control Objectives

3.1.1 Safety: Insuring the invariance of a set of states $\mathcal{O}$

Let $\mathcal{O}_{S_c}$ be the orbit[11] of the controlled system. The problem is to compute (C, C_0) such that $\mathcal{O}_{S_c} \subseteq \mathcal{O}$. Assume that there exists an acceptable controller (C, C_0) which satisfies this property. In this case, we know that:

1. $\mathcal{O}_{S_c} \subseteq \mathcal{O}$
2. $\mathcal{O}_{S_c}$ invariant for S_c, which means that

$$\forall x \in \mathcal{O}_{S_c}, \forall y \in (\mathbb{Z}/3\mathbb{Z})^m, \forall u \in (\mathbb{Z}/3\mathbb{Z})^p,\ \{Q(x,y,u) = 0 \text{ and } C(x,y,u) = 0\} \Rightarrow P(x,y,u) \in \mathcal{O}_{S_c}.$$

Now let x be a state of $\mathcal{O}_{S_c}$ and y an event admissible in x by the system S. Since (C, C_0) is an acceptable controller, y is also admissible in x for the controlled system S_c. Then, there exists a control u such that $Q(x,y,u) = 0$ and $C(x,y,u) = 0$, and, for this control, $P(x,y,u) \in \mathcal{O}_{S_c}$. According to Definition 7, $\mathcal{O}_{S_c}$ is *U-invariant under control* for the system S.

If there exists an acceptable controller which ensures the invariance of $\mathcal{O}$, then the orbit $\mathcal{O}_{S_c}$ of S_c is included in $\mathcal{O}$ and is U-invariant under control for S. It is also necessary to have some initial states of S included in $\mathcal{O}$. In fact, these three properties are sufficient:

Theorem 6 *Given a controllable system S and a set of states $\mathcal{O}$ of S, there exists an acceptable controller which ensures the invariance of $\mathcal{O}$ if and only if there exists a set of states E such that:*

(α1) $E \subseteq \mathcal{O}$,

(α2) $E \cap \mathcal{V}(<Q_0>) \neq \emptyset$, *where* $\mathcal{V}(<Q_0>) = \{x / Q_0(x) = 0\}$

(α3) E *is U-invariant under control for* S.

Proof: We have already shown that the conditions are necessary. Conversely, assume there exists a set E which satisfies the three properties (α1), (α2) and (α3). Let (C, C_0) be such that $< C_0 >= \mathcal{I}(E)$ and $C = P^\star(C_0)$. By construction,

[11] We recall that the orbit of a PDS S is the set of states that are reachable from one of the initial states.

- $C_0(x) = 0 \quad \Leftrightarrow \quad x \in E,$
- $C(x, y, u) = 0 \quad \Leftrightarrow \quad P(x, y, u) \in E.$

It follows that the orbit of S_c is included in E and hence in $\mathcal{O}$; the controller (C, C_0) ensures the invariance of $\mathcal{O}$. Now, let x be a state in the orbit of S_c and let y be an event admissible in x for S. x is also an element of E and since E is U-invariant under control, there exists a control u such that $Q(x, y, u) = 0$ and $P(x, y, u) \in E$. This is equivalent to $Q(x, y, u) = 0$ and $C(x, y, u) = 0$. y is then also admissible in x for S_c. Since the condition $E \cap \mathcal{V}(<Q_0>) \neq \emptyset$ means that $C_0(X) = 0$ and $Q_0(X) = 0$ have common solutions, the controller (C, C_0) is acceptable. $\diamond$

The proof gives an algorithm to compute a controller ensuring the invariance of a set of states $\mathcal{O}$. It suffices to find the largest U-invariant under control subset of $\mathcal{O}$ which satisfies the initialization condition $(\alpha 2)$. Let E be this subset; if E satisfies the condition $(\alpha 2)$ then control equations can be obtained from the principal generator of E, otherwise no subset of $\mathcal{O}$ can satisfy condition $(\alpha 2)$ and the problem has no solution.

From 2, F is U-invariant under control if and only if $F \subseteq \widetilde{pre}\,(F)$. The largest U-invariant under control subset of $\mathcal{O}$ is obtained by constructing the sequence $(E_i)_{i \in \mathbb{N}}$ defined by:

$$\left\{ \begin{array}{lcl} E_0 & = & \mathcal{O} \\ E_{i+1} & = & E_i \cap \widetilde{pre}\,(E_i). \end{array} \right.$$

The sequence is decreasing. Since all sets E_i are finite, there exists an index j such that $E_{j+1} = E_j$. The set E_j is the largest U-invariant under control subset of $\mathcal{O}$. In practice, we transform this computation into an equivalent one over the ideals $(\underline{a}_i)_{i \in \mathbb{N}} = \mathcal{I}(E_i)_{i \in \mathbb{N}}$ associated with the set E_j.

$$\left\{ \begin{array}{lcl} \underline{a}_0 & = & \mathcal{I}(E) \\ \underline{a}_{i+1} & = & \underline{a}_i + \forall elim_Y(< Q' >^c \cap \exists elim_U(< Q > + P^*(\underline{a}_{i+1}))) \end{array} \right.$$

where $< Q' >^c$ denotes the ideal associated with the complementary set of $\mathcal{V}(< Q' >) = \exists elim_U(\mathcal{V}(< Q >))$.

3.1.2 Safety + Reachability

We can also consider control objectives that are conjunction of basic properties of state trajectories. However, basic properties cannot, in general, be combined in a modular way. For example, a safety property puts restrictions on the set of state trajectories which may be not compatible with an reachability property. The synthesis of a controller insuring both properties must be effected by considering both properties *simultaneously* and not by combining a controller insuring safety with a controller insuring reachability independently.

Let us denote by $\mathcal{O}_1$ the set of states representing the safety property and $\mathcal{O}_2$, the set of states representing the reachability property. The "safe" reachability we have in mind is the following: the controller we are looking for, ensures that all the trajectories of the controlled system are included in $\mathcal{O}_1$ while for each point of the orbit there exists a safe trajectory starting in this point that reaches $\mathcal{O}_2$.

Suppose there exists such a controller and let $\mathcal{O}'_{S_c}$ be the orbit of S_c. $\mathcal{O}'_{S_c}$ satisfies the same three conditions as previously and a supplementary one: any state $x \in \mathcal{O}'_{S_c}$ is the origin of a trajectory in S_c which reaches $\mathcal{O}_2$. But any trajectory $x_0 \xrightarrow{y_0} x_1 \xrightarrow{y_1} \ldots \xrightarrow{y_{k-1}} x_k$ of S_c is also a trajectory of S where all the visited states belongs to $\mathcal{O}'_{S_c}$.

The sequence $(D_i)_{i\in\mathbb{N}}$ defined by

$$\left\{ \begin{array}{rcl} D_0 & = & E \cap \mathcal{O}_2 \\ D_{i+1} & = & D_i \cup (E \cap pre(D_i)), \end{array} \right. \tag{25}$$

converges whatever the set E is. The fixed point is the set of states which are origins of a trajectory of S leading to $\mathcal{O}_2$ and containing only states of E. This set is noted $Reach(E, \mathcal{O}_2)$. The supplementary property of the orbit $\mathcal{O}'_{S_c}$ can now be rewritten $\mathcal{O}'_{S_c} \subseteq Reach(\mathcal{O}'_{S_c}, \mathcal{O}_2)$.

Theorem 7 *Given a controllable system S, and two set of states $\mathcal{O}_1$ and $\mathcal{O}_2$, there exists an acceptable controller which guarantees the invariance of $\mathcal{O}_1$ and the reachability of $\mathcal{O}_2$ if and only if there exists a set of states $\mathcal{O}'_{S_c}$ such that:*

(β_1) $\mathcal{O}'_{S_c} \subseteq \mathcal{O}_1$,

(β_2) $\mathcal{O}'_{S_c} \cap \mathcal{V}(<Q_0>) \neq \emptyset$,

(β_3) $\mathcal{O}'_{S_c}$ *is U-invariant under control for* S,

(β_4) $\mathcal{O}'_{S_c} \subseteq Reach(\mathcal{O}'_{S_c}, \mathcal{O}_2)$.

Proof: The proof is similar to the one of Theorem 6. If such an $\mathcal{O}'_{S_c}$ exists, let C_0 be such that $< C_0 >= \mathcal{I}(\mathcal{O}'_{S_c})$ and $C = P^\star(C_0)$. From Theorem 6, the pair (C, C_0) is an acceptable controller ensuring the invariance of $\mathcal{O}_1$ (properties (β_1), (β_2) and (β_3)). Then we just need to prove that the controller (C, C_0) ensures the reachability of $\mathcal{O}_2$. To do so, let x be an element of $\mathcal{O}'_{S_c}$, since x belongs to $Reach(\mathcal{O}'_{S_c}, \mathcal{O}_2)$ (thanks to ($\alpha 4$), there exists a trajectory of S of the form: $x_0 \xrightarrow{y_0,u_0} x_1 \xrightarrow{y_1,u_1} \ldots \xrightarrow{y_{k-1},u_{k-1}} x_k$, with $x_0 = x$, $x_k \in \mathcal{O}_2$ and $\forall i, 0 \leq i \leq k, x_i \in \mathcal{O}'_{S_c}$. For all the indices $i \in [0 \ldots k-1]$, we have $Q(x_i, y_i, u_i) = 0$ and $P(x_i, y_i, u_i) \in \mathcal{O}'_{S_c}$. This is equivalent to say that $Q(x_i, y_i, u_i) = 0$ and $C(x_i, y_i, u_i) = 0$, which shows that the trajectory is also a trajectory of S_c starting from x and reaching $\mathcal{O}_2$. Hence (C, C_0) ensures the reachability condition. ◇

The controller computation follows the same principle as in the invariance case. It consists in finding the greatest U-invariant under control subset $\mathcal{O}'_{S_c}$ of $\mathcal{O}_1$ which also satisfies $\mathcal{O}'_{S_c} \subseteq Reach(\mathcal{O}'_{S_c}, \mathcal{O}_2)$.

This is obtained by constructing the sequence of set of states $(E_i)_{i\in\mathbb{N}}$ as follows:

$$\left\{\begin{array}{rcl} E_0 & = & \mathcal{O}_1 \\ E_{i+1} & = & Reach(E_i \cap \overset{\approx}{pre}\,(E_i), \mathcal{O}_2). \end{array}\right.$$

By construction of *Reach*, for any set E, $Reach(E, \mathcal{O}_2)$ is included in E; it implies that $E_{i+1} \subseteq E_i$. The sequence is decreasing and because the sets E_i's are finite, there exists j such that $E_j = E_{j+1}$. It follows three inclusions: $E_j \subseteq \mathcal{O}_1$, $E_j \subseteq \overset{\approx}{pre}\,(E_j)$ and $E_j \subseteq Reach(E_j, \mathcal{O}_2)$. Then, E_j is U-invariant under control (by Corollary 2), E_j is included in $\mathcal{O}_1$ and satisfies condition (β_4). It can also be shown that E_j is the greatest such set. Like previously, all these computations can be rephrased in terms of ideals.

3.2 Partial Order Control Problem

This section deals with control objectives that consider the way to reach a given logical goal. Sometimes, these control objectives can be expressed as partial order relations between the states of the system.

3.2.1 General method

Let S be a CPDS as in (20). Let us suppose that the system evolves into a state x, and that an event y is admissible in x. Because in general, the system is nondeterministic, there may be have several controls u such that $Q(x, y, u) = 0$. Let u_1 and u_2 be two controls compatible with y in x. Then the system can evolve into either $P(x, y, u_1)$ or $P(x, y, u_2)$. Our goal is to synthesize a controller that will choose between u_1 and u_2, in such a way that the system evolves into the "best" state according to a choice criterion.

Order Relations: To capture this criterion, we introduce a (partial) quasi-order relation (called order relation in the sequel) over the states of the system, noted $\succeq$. We will note $x \not\succeq x'$, to express that two states x and x' are not comparable (*i.e.*, if neither $x \succeq x'$ nor $x' \succeq x$).

Remark 4 *For some order relations we will consider that one of the two following situations (or both) can hold for two given states of the state space:*

1. $x \succeq x'$ *and* $x' \succeq x \not\Rightarrow x = x'$ *for some* x, x' (i.e., *the order relation is not antisymmetric).*
2. $x \not\succeq x'$ *and* $x' \not\succeq x$ *for some* x, x'. (i.e., *the order relation can be partial).* $\diamond$

Since the set of states is finite, each (partial) order relation $\succeq$ can be translated into a polynomial equation of $A[X, X']$ such as: $R_{\succeq}(x, x') = 0 \Leftrightarrow x \succeq x'$. As we deal with a (non strict)

order relation, from $\succeq$ (and then $R_{\succeq}$) we construct a strict (partial) order relation $\succ$ defined as:

$$x \succ x' \Leftrightarrow \{x \succeq x' \wedge \neg(x' \succeq x)\} \tag{26}$$

Or equivalently, $x \succ x' \Leftrightarrow R_{\succ}(x, x') = 0$, where $R_{\succ}(X, X') = R_{\succeq}(X, X') \oplus (1 - R^2_{\succeq}(X', X))$

Controller synthesis: We now introduce the control policy we want to be applied on the system.

Definition 9 *With the preceding notations, given an order relation $\succeq$, a CPDS S is said to satisfy the partial order relation $\succeq$, if for each admissible pair (x, y), S evolves into a state that is maximal for the order relation $\succ$.* •

Without control, the system can start from one of the initial states of $\mathcal{V}(< Q_0 >) = \{x \ / \ Q_0(x) = 0\}$. The new possible initial states are the maximal states (for $R_{\succ}$) among all the solutions of the equation $Q_0(X) = 0$. Their set, say I_0, is obtained by removing from $\mathcal{V}(< Q_0 >)$ all the states for which there exist at least one smaller state for the strict order relation $\succ$, i.e.

$$I_0 = \mathcal{V}(< Q_0 >) - \{x \ / \ \exists x' \in \mathcal{V}(< Q_0 >), x' \succ x\}. \tag{27}$$

Proposition 10 *The principal generator of I_0 is*

$$Q_0(X) \oplus \{1 - \exists elim_{X'}(Q_0(X') \oplus R_{\succ}(X, X'))\}. \tag{28}$$

This polynomial will constitute the C_0 component of the controller. ◇

Following this transformation, we only keep in I_0 the maximal elements of the relation $\succ$, but also those that are not comparable with any other state. Note that if we had considered $\succeq$ instead of $\succ$ to compute I_0, we would have also lost the states satisfying the item *1.* of Remark 4.

We are now interested in the control policy, *i.e.*, how to choose the right control when the system S is into current state x and when an event y occurs.

Definition 10 *Let us consider u_1 and u_2 two controls compatible with the event y in the state x, then the control u_1 is said to be* better *than the control u_2, if $P(x, y, u_1) \succ P(x, y, u_2)$. Or equivalently, $R_{\succ}(P(x, y, u_1), P(x, y, u_2)) = 0$.* •

To do so, let us introduce a new (partial) order relation $\sqsupset$ derived from $\succ$.

$$(x, y, u) \sqsupset (x', y', u') \Leftrightarrow \begin{cases} x = x' \\ y = y' \\ P(x, y, u) \succ P(x, y, u') \end{cases} \tag{29}$$

In other words a triple (x, y, u) is "better" than a triple (x, y, u') whenever the state $P(x, y, u)$ reached by choosing the control u is better than the state $P(x, y, u')$ reached by choosing the control u'.

As for the initial set of states, we compute the maximal triples of this new order relation among all of the triples. To this effect, we first define $J = \{(x, y, u) \in (\mathbb{Z}/_{3\mathbb{Z}})^{n+m+p} \ / \ Q(x, y, u) = 0\}$, the set of admissible triples (x, y, u). The maximal set of triples, say J_{max}, is then

$$J_{max} = J - \{(x, y, u) \ / \ \exists (x, y, u') \in J, (x, y, u') \sqsupseteq (x, y, u)\}. \tag{30}$$

Using algebraic relations of Section 1.2, we obtain

Proposition 11 *The principal generator of J_{max} is*

$$Q(X, Y, U) \oplus (1 - \exists elim_{U'}(Q(X, Y, U') \oplus R_{succ}(P(X, Y, U'), P(X, Y, U)))) \tag{31}$$

This polynomial will constitute the C component of the controller. ◇

Using this controller, the choice of a control u, compatible with y in x, is constrained is such a way that the possible successor state is maximal for $\succ$.

Theorem 8 *With the preceding notations, the controller (C, C_0) is an acceptable controller for the CPDS S defined in (20). Moreover, the controlled system $S_C = S \oplus (C, C_0)$ fulfills the control policy of Definition 9.*

Proof: We first prove that the controller (C, C_0) is an acceptable controller with respect to the CPDS S. By construction, $I_0 = \mathcal{V}(< C_0 >) \subseteq \mathcal{V}(< Q_0 >)$. Moreover, assume that the system S is in a state x and that y is an admissible event for this system. Among all the controls compatible with y in x (there exists at least one since y is admissible), there exists a triple (x, y, u) that is maximal for the (partial) order relation $\sqsupset$; it then satisfies $C(x, u, u) = 0$. Then $C(x, u, u) = 0$ and $Q(x, y, u) = 0$, which entails that y is also admissible for S_C. Assume now that the system S_C is in a state x and that the event y occurs. Let u be an admissible control for (x, y). This means that $Q(x, y, u) = 0$ and $C(x, u, u) = 0$. In other words, the triple (x, y, u) is maximal for $\sqsupset$, and then, among all the controls compatible with y in x, u is such that $P(x, y, u)$ is maximal for $\succ$. ◇

3.2.2 Examples of order relations

Different kinds of order relations (or partial order relations) can be used to express properties over states.

Minimally restrictive constraints on uncontrollable events: Let us assume that the system S is in a state x and that it receives the event y; then the system can choose any control such that $Q(x,y,u)=0$. Let u_1 and u_2 be two controls compatible with y in x. Let us write $x_1=P(x,y,u_1)$ and $x_2=P(x,y,u_2)$. Consider Ad_1 and Ad_2, the sets of admissible y events in, respectively, x_1 and x_2. Using $Q'(X,Y)=\exists elim_U(Q(X,Y,U))$, we have

$$\left\{\begin{array}{lcl} Ad_1 & = & \{y\in(\mathbb{Z}/3\mathbb{Z})^m \ / \ Q'(x_1,y)=0\} \\ Ad_2 & = & \{y\in(\mathbb{Z}/3\mathbb{Z})^m \ / \ Q'(x_2,y)=0\} \end{array}\right.$$

Definition 11 *With the preceding notations, x_1 is said to be less restrictive than x_2 (noted $x_1\succeq x_2$) if and only if $Ad_2\subseteq Ad_1$, which in turn can be easily proved to*

$$\forall y\in(\mathbb{Z}/3\mathbb{Z})^m,\ Q'(x_2,y)=0\Rightarrow Q'(x_1,y)=0.$$

•

$Ad_2\subseteq Ad_1$ means that there is more spontaneous evolution in x_1 than in x_2. We then want the controller to choose the control u_1 rather than the control u_2. Now, a polynomial expression for $\succeq$ of Definition 11 can be derived:

Proposition 12 $x\succeq x'\Leftrightarrow R_{\succeq}(x,x')=0$, *where* $R_{\succeq}(X,X')=\forall elim_Y((1-Q'^2(X',Y))Q'(X,Y))$.

◇

Applying the methods described in Section 3.2.1, leads to synthesize a controller such that the controlled system respects the control strategy of minimally restrictive constraints on uncontrollable events.

Maximization of the number of state variables equal to 1 Let $(X_1,\ldots,X_n)$ be the set of state variables X, where the integer n represents the number of state variables.

Definition 12 *let $x=(x_1,\ldots,x_n)$ and $x'=(x'_1,\ldots,x'_n)$ be two states, then we say $x_1\sqsupseteq x_2$ if and only if $\forall i\in[1..n],\ x_i=1\Rightarrow x'_i=1$.* •

To express this partial order relation, we need to introduce the polynomial function δ from $(\mathbb{Z}/3\mathbb{Z})\times(\mathbb{Z}/3\mathbb{Z})$ to $\mathbb{Z}/3\mathbb{Z}$ such that

$$\delta(a,b)=(a(a+1)(1-b))^2\ . \tag{32}$$

It is straightforward to show that $\delta(a,b)=0\Leftrightarrow\{(a=1)\Rightarrow(b=1)\}$. The previous partial order relation expressed by the Definition 12, can then be translated in polynomial terms:

Proposition 13 $x\sqsupseteq x'$, *iff* $R_{\sqsupseteq}(x,x')=0$ *with* $R_{\sqsupseteq}(X,X')=\bigoplus_{i=1}^{n}\delta(X_i,X'_i)$

Proof: Assume that $x\sqsupseteq x'$, then $\forall i\in[1..n],\ x_i=1\Rightarrow x'_i=1$. This is equivalent to say that $\forall i\in[1..n],\delta(x_i,x'_i)=0$ and so $\bigoplus_{i=1}^{n}\delta(x_i,x'_i)=0$. The converse is similar. ◇

By applying the construction described in Section 3.2.1, it is possible to synthesize a controller which chooses, in a state x, one of the best controls for the relation $R_{\sqsupseteq}$. In other words, such a control leads the system in a state where the number of state variables equal to 1 is maximal.

Though it is always possible to express priorities over the states using algebraic order relations (since the set of states is finite), it is sometimes more natural to express directly the priorities using numerical cost functions.

Numerical order relations: We here use cost functions over states to express order relations. Let $X = (X_1, \ldots, X_n)$ be the state variables of the system. Then, a cost function is a mapping from $(\mathbb{Z}/3\mathbb{Z})^n$ to $\mathbb{N}$, which associates to each state x of $(\mathbb{Z}/3\mathbb{Z})^n$ its cost.

Definition 13 *Given a CPDS S and a cost function c over the states of the system. A state x_1 is said to be c-better than a state x_2 (denoted by $x_1 \succeq_c x_2$), if and only if, $c(x_2) \geq c(x_1)$.* •

In order to express the quasi-order relation $\succeq_c$ as a polynomial relation, let us consider

$$k_{max} \ = \ sup_{x \in (\mathbb{Z}/3\mathbb{Z})^n}(c(x)).$$

Now, for all $i \in [0, ..k_{max}]$, the following sets of states are computed

$$A_i = \{x \in (\mathbb{Z}/3\mathbb{Z})^n \ / \ c(x) \ = \ i\}. \tag{33}$$

These sets $(A_i)_{i=0..k_{max}}$ form a partition of the state space. Note that some A_i can be reduced to the empty set. The proof of the following property is easy

Proposition 14 *$x_1 \succeq_c x_2$ if and only if $\exists i \in [0, .., k_{max}],\ x_1 \in A_i \wedge x_2 \in \bigcup_{j=i}^{k_{max}} A_j$* ◇

Let $g_0, \ldots, g_{k_{max}}$ be the principal generators of the ideals associated to the sets $A_1, \ldots, A_{k_{max}}$[12]. Then,

Proposition 15 *$x \succeq_c x'$ iff $R_{\succeq_c}(x, x') = 0$, where*

$$R_{\succeq_c}(X, X') = \prod_{i=0}^{k_{max}} \{g_i^2(X) \oplus (\prod_{j=i}^{k_{max}} (g_j^2(X')))\} \tag{34}$$

◇

Again, as this order relation $\succeq_c$ is now expressed as a polynomial relation, we are able to use the method described in Section 3.2.1 to synthesize the corresponding controller.

Some other order relations can be considered. Among them, we can mention the *maximization of the number of state variables equal to 1* [54], or the *stabilization of a system*. Finally note that the notion of numerical order relation has been generalized over a bounded states trajectory of the system, retrieving the notion of *Optimal Control* [53][13].

[12] To compute efficiently such principal generators, it is important to use on the Arithmetic Decision Diagrams (ADD) developed, for example, by [16].

[13] Some other similar works can be found in in [37, 61, 68, 69, 50].

4 Dynamical controller synthesis

Many properties of discrete event system cannot be tackled with static relations. For example, we cannot express that an event y never to have the same values consecutively, i.e. to satisfy the relation $\forall t,\ y_{t+1} - y_t \neq 0$. Such relations are called relations of order k when they involve time indexes from t to $t+k$, or k-locally testable properties. These kinds of properties lead to dynamical controllers. The main idea is to proceed to the k extension of the initial system so that the initial control objective reduces to a static control objective for the extended system (defined below).

So, let us consider a k-locally testable property, that is of the form:

$$K(X_{t+k}, \ldots, X_t, Y_{t+k}, \ldots, Y_t, U_{t+k}, \ldots, U_t) = 0 \tag{35}$$

The k-extension S_{new} of the CPDS S w.r.t. the k-locally testable property K is built as follows:

- the states of S_{new} are of the form $X^t = (X_{t+k}, \ldots, X_t, Y_{t+k}, \ldots, Y_t, U_{t+k}, \ldots, U_t)$.
- The uncontrollable events are of the form $Y^t = (X_{t+k+1}, Y_{t+k+1})$;
- The controls are simply given by the controls of the initial system at the instant $t+k+1$: $U^t = U_{t+k+1}$.

The obtained CPDS is then the following:

$$S_{new} : \left\{ \begin{array}{rcl} X^{t+1} & = & P_{new}(X^t, Y^t, U^t) \\ Q_{new}(X^t, Y^t, U^t) & = & 0 \\ Q^0_{new}(X^0) & = & 0 \end{array} \right. \tag{36}$$

where P_{new}, Q_{new} and Q^0_{new} are defined by:

$$\left\{ \begin{array}{l} P_{new}(X^t, Y^t, U^t) = (X_{t+k+1}, .., X_{t+1}, Y_{t+k+1}, .., Y_{t+1}, U_{t+k+1}, .., U_{t+1}) \\ \\ Q_{new}(X^t, Y^t, U^t) = Q(X_{n+k+1}, Y_{t+k+1} + U_{t+k+1}) \oplus \\ \qquad\qquad [\oplus_{i=0}^{k} \{(X_{t+i+1} - P(X_{t+i}, Y_{t+i}, X_{t+i})) \oplus Q(X_{t+i}, Y_{t+i}, U_{n+i})\}] \\ \\ Q^0_{new}(X^0) = Q_0(X_0) \oplus Q(X_0, Y_0, U_0) \oplus (X_1 - P(X_0, Y_0, U_0)) \oplus \ldots \oplus \\ \qquad\qquad Q(X_k, Y_k, U_k) \oplus (X_k - P(X_{k-1}, Y_{k-1}, U_{k-1})) \end{array} \right.$$

It can be shown that for any trajectory $(x^t, y^t, u^t)_{i \in \mathbb{N}}$ of S_{new}, is a trajectory of S. Now, to use the techniques of Section 2 in order to obtain a controller,we show that a solution for S_{new} is indeed a solution for S. In order to make the discussion more concrete, we consider the case of a safety control objective expressed as a k-locally testable property.

Assume that there exists a controller (C_{new}, C^0_{new}) that can solve this control objective for the extended system S_{new}. We must show that this controller is acceptable for S: according to Definition 8, such a controller must not add new constraints on the uncontrollable events of S. Assume the system is in a state x_t, with $t > k$. If y_t is an admissible event in x_t for S, there exists a control u_t such that $Q(x_t, y_t, u_t) = 0$. Taking into account the predecessors of (x_t, y_t, u_t), we consider the state x^{t-k-1} of S_{new}. Then $y^{t-k-1} = (x_t, y_t)$ certainly constitutes an admissible event in x^{t-k-1} for S_{new}, and therefore (x_t, y_t) for the controller of S_{new} since this controller is acceptable. This shows the first condition for (C_{new}, C^0_{new}) to be acceptable for S.

The second condition concerns the initial states. A simple method to implement the initialization phase is to successively add constraints on S.

$$< C^0_{init}(X_0) >=< C^0_{new} > \cap \mathbb{Z}/_{p\mathbb{Z}}\,[X_0]$$

$$< C^1_{init}(X_0, Y_0, U_0) >=< C^0_{new} > \cap \mathbb{Z}/_{p\mathbb{Z}}\,[X_0, Y_0, U_0]$$
$$\ldots$$
$$< C^k_{init}(X_0, Y_0, U_0, .., X_k, Y_k, U_k) >=< C^0_{new} > \cap \mathbb{Z}/_{p\mathbb{Z}}\,[X_0, Y_0, U_0, .., X_k, Y_k, U_k]$$

A trajectory $(x_t, y_t, u_t)_{t \leq k}$ that satisfies these constraints will certainly satisfy the initial constraints of S_{new}. Moreover, if the initial conditions of the controller for S_{new} are compatible with the initial conditions of S, such a trajectory necessarily exists. However, it is not sufficient to obtain an acceptable controller since it is forbidden to add new constraints on the uncontrollable events, even during the initialization phase. In fact, every successive events $y_0, y_1, \ldots, y_k$ admissible in S have to be admissible for the initial constraints of the controller. Translated into ideals leads to:

$$C^0_{new}(X^0) \cap \mathbb{Z}/_{p\mathbb{Z}}\,[Y_o, \ldots, Y_k] \subseteq Q_{new} \cap \mathbb{Z}/_{p\mathbb{Z}}\,[Y_o, \ldots, Y_k]$$

This methodology solve the k-locally testable control problem. In the same time, we obtain a method that solves the controller synthesis problem for properties that are concerned by both states and events.

5 Integration in the Signal environment

We present a tool developed in the SIGNAL environment allowing the visualization of the synthesized controller by interactive simulation of the controlled system [51].

5.1 The SIGNAL environment

The technique involved for the integration is the *synchronous approach* to reactive real time systems [12]. One way of interpreting the synchronous hypothesis consists in considering that computations produces values that are relevant whitin a single logical instant of time. A

family of languages is based on this hypothesis [29] among them ESTEREL [14], LUSTRE[30] and SIGNAL. Here, to specify our model, we will use the synchronous data flow language SIGNAL [9, 43, 24].

The aim of SIGNAL is to support the design of safety critical applications, especially those involving signal processing and process control. The synchronous approach [29] guarantees the determinism of the specified systems, and supports techniques for the detection of causality cycles and logical incoherences. The design environment features a block-diagram graphical interface, a formal verification tool SIGALI, and a compiler that computes a hierarchy of inclusion of logical clocks (representing the temporal characteristics of discrete events), checks for the consistency of the inter-dependencies, and automatically generates optimized executable code ready to be embedded in environments for simulation, test, prototyping or the system itself.

5.1.1 The SIGNAL language.

The SIGNAL language [43] manipulates *signals* `X`, which denote unbounded series of typed values (flows), indexed by time. An associated *clock* determines the set of instants at which values are present. The constructs of the language can be used in an equational style to specify the relations between signals, *i.e.*, between their values and between their clocks. Data flow applications are activities executed over a set of instants in time. At each instant, input data is acquired from the execution environment; output values are produced according to the system of equations considered as a network of operations.

The SIGNAL language is defined by a small kernel of operators. Each operator has formally defined semantics and is used to obtain a clock equation and the data dependencies of the participating signals.

Functions are instantaneous transformations on the data. The definition of a signal Y_t by the function f: $\forall t$, $Y_t = f(X_{1_t}, X_{2_t}, \ldots, X_{n_t})$ is written in SIGNAL: `Y := f{ X1, X2, ... , Xn}`. The signals `Y`, `X1`, ... , `Xn` are required to have the same clock.

Selection of a signal `X` according to a boolean condition `C` is: `Y := X when C`. If `C` is present and *true*, then `Y` has the presence and value of `X`. The operands and the result do not have identical clock. The clock of `Y` is the *intersection* of that of `X` and that of `C` at the value *true*.

Deterministic merge noted: `Z := X default Y` has the value of `X` when it is present, or otherwise that of `Y` if it is present and `X` is not. Its clock is the *union* of that of `X` and that of `Y`.

Delay gives access to past values of a signal. E.g., the equation $ZX_t = X_{t-1}$, with initial value V_0 defines a *dynamic process*. It is encoded by: `ZX := X$1` with initialization `ZX init V0`. `X` and `ZX` have equal clocks.

Parallel Composition of processes is made by the associative and commutative operator " | ", denoting the union of the underlying systems of equations. Systems communicate and interact through signals defined in one system and featured in others. For these signals,

composition preserves constraints from all systems, especially temporal ones. This means that they are present if the equations systems allow it. In Signal , for processes `P1` and `P2` , composition is written: `(| P1 | P2 |)`

The following table illustrates each of the primitives with a trace:

n	3	2	1	0	3	2	...
zn := n$ 1 init 0	0	3	2	1	0	3	...
p := zn-1	-1	2	1	0	-1	2	...
x := true when (zn=0)	t				t		
y := true when (n=0) default (not x)	f			t	f		

The rest of the language is built upon this kernel. A structuring mechanism is proposed in the form of process schemes, defined by a name, typed parameters, input and output signals, a body, and local declarations. Occurrences of process schemes in a program are expanded (like macro-expansion) by a pre-processor of the compiler. Derived processes have been defined on the base of the primitive operators, providing programming comfort. E.g., the instruction `X ^= Y` specifies that signals `X` and `Y` are synchronous (i.e., have equal clocks); `when B` gives the clock of *true*-valued occurrences of `B`. Arrays of signals and of processes have been introduced as well. Hierarchy, modularity and re-use of the definition of processes are supported by the possibility of defining process models that can be invoked by instantiation. See [43] for a more detailed description.

5.2 The graphical interface

The complete programming environment of SIGNAL also contains a graphical, block-diagram oriented user interface [13], as illustrated in Figure (2). This environment allows the user to

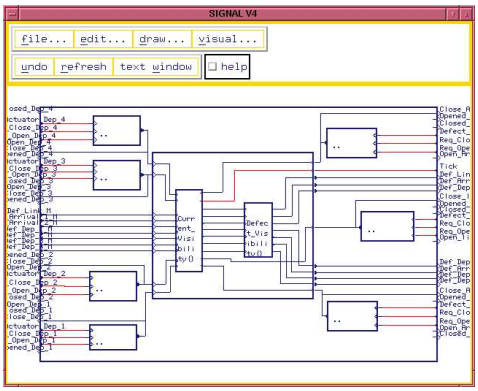


Figure 2: An example of SIGNAL process using the graphical interface

have graphical and textual representations of the language structures. These representations may be used together during the building or the "reading" of the program.

A SIGNAL expression may be considered as a set of components (represented as boxes) with connection points (ports represented with triangles) joined by links. The same SIGNAL expression is also a system of equations on series of values; this system is represented by a phrase built with the language operators (definitions of variables, composition, renaming, etc.)

This graphical interface will be used further in order to perform the specification of the system to be controlled as well as the automatic simulation of the controlled system.

5.2.1 SIGALI: The formal proof system

The SIGNAL environment contains a verification and controller synthesis tool-box, SIGALI. This tool allows us to prove the correctness of the dynamical behavior of the system. The equational nature of the SIGNAL language leads to the use of polynomial dynamical equation systems (PDS) over $\mathbb{Z}/3\mathbb{Z}$ as a formal model of program behavior.

Logical abstraction of a SIGNAL program: To model its behavior, a SIGNAL process is translated into a system of polynomial equations over $\mathbb{Z}/3\mathbb{Z}$ [42]. Note that this translation is automatically performed by the SIGNAL compiler.

Signals. The three possible states of a boolean signal `X` (*i.e.*, *present* and *true*, *present* and *false*, or *absent*) are coded in a *signal variable* x by:

$$\begin{cases} present \wedge true & \rightarrow & +1 \\ present \wedge false & \rightarrow & -1 \\ absent & \rightarrow & 0 \end{cases}$$

For the non-boolean signals, , as we are only interested in the logical part of the SIGNAL program, we only code the fact that the signal is *present* or *absent*[14]:

$$\begin{cases} present & \rightarrow & +1 \\ absent & \rightarrow & 0 \end{cases}$$

Note that the square of *present* is 1, whatever its value, when it is present. Hence, for a signal `X`, its clock can be coded by x^2. It follows that two synchronous signals `X` and `Y` satisfy the constraint equation: $x^2 = y^2$. This fact is used extensively in the following.

Primitive processes. Each of the primitive processes of SIGNAL are then encoded as polynomial equations. Let us just consider the example of the *selection* operator. `C := A when`

[14] In other words, we just translate the synchronization between the signals.

`B` means "*if* $b = 1$ *then* $c = a$ *else* $c = 0$". It can be rewritten as a polynomial equation: $c = a(-b - b^2)$. Indeed, the solutions of this equation are the set of possible behaviors of the primitive process `when`. For example, if the signal B is *true* (*i.e.*, b=1),then $(-b - b^2) = (-1 - 1) = 1$ in $\mathbb{Z}/3\mathbb{Z}$, which leads to $c = a$.

The delay `$`, which is dynamical, is different because it requires memorizing the past value of the signal into a *state variable* x. In order to encode `B := A$1 init B0`, we have to introduce the three following equations:

$$\begin{cases} x' &= a + (1 - a^2)x & (1) \\ b &= xa^2 & (2) \\ x_0 &= b_0 & (3) \end{cases}$$

where x' is the value of the memory at the next instant. Equation (1) describes what will be the next value x' of the state variable. If a is *present*, x' is equal to a (because $(1 - a^2) = 0$), otherwise x' is equal to the last value of a, memorized by x. Equation (2) gives to b the last value of a (*i.e.* the value of x) and constrains the clocks b and a to be equal.Equation (3) corresponds to the initial value of x, which is the initial value of b.

Table 1 shows how all the primitive operators are translated into polynomial equations:

Boolean expressions	
`B := not A`	$b = -a$
`C := A and B`	$c = ab(ab - a - b - 1)$ $a^2 = b^2 = c^2$
`C := A or B`	$c = ab(1 - a - b - ab)$ $a^2 = b^2 = c^2$
`C := A default B`	$c = a + (1 - a^2)b$
`C := A when B`	$c = a(-b - b^2)$
`B := A $1 (init` b_0`)`	$x' = a + (1 - a^2)x$ $b = a^2x$ $x_0 =$ `b`$_0$
non-boolean expressions	
`B := `$f(A_1, \ldots, A_n)$	$b^2 = a_1^2 = \cdots = a_n^2$
`C := A default B`	$c^2 = a^2 + b^2 - a^2b^2$
`C := A when B`	$c^2 = a^2(-b - b^2)$
`B := A $1 (init` b_0`)`	$b^2 = a^2$

Table 1: Translation of the primitive operators.

Processes. Using this method, any SIGNAL specification can then be translated into a set of equations called polynomial dynamical system (PDS) as the one presented in Section 1.1.

$$S = \begin{cases} X' &= P(X, Y, U) \\ Q(X, Y, U) &= 0 \\ Q_0(X) &= 0 \end{cases} \quad (37)$$

For example the following small process in SIGNAL,

```
process alternate = {?  event A,B !}
   (| C := not ZC
    | ZC := X$1
    | synchro{A,when C}
    | synchro{B,when ZC}
    |)
  where
    logical C, ZC init false
end
```

is translated in the polynomial dynamical system with variables a, b, x and zx corresponding to the events A, B and the logical signals X and ZX and a state variable ξ introduced by the delay. The system consists of

- an initialization equation : $x = -1$,
- an evolution equation : $x' = x + (1 - c^2) * x$
- and a system of constraint equations

$$c = -zc, zc = x * c^2, a^2 = when\ c = -c - c^2, b^2 = when\ zc = -zc - zc^2$$

we use the symbol `when` in order to simplify the presentation: it can be encoded as shown in Table 1.

Once obtained this polynomial dynamical system, and using algebraic methods described in the previous section, we are able to compute controllers (C, C_0) which ensure the various control objectives detailed in the previous sections.

- "Traditional" control objectives such that:
  - the *reachability* of a set of states from the initial states of the system ,
  - the *attractivity* of a set of states E from a set of states F.
  - *persistence* of a set of states E.
  - *reccurence* of a set of states E.
  - *safety + { reachability, attractivity, persistence}* [23].
- Control objectives expressed as partial order relations over the states of the polynomial dynamical system [52] such that:

- the *minimally restrictive control* (choice of a command such that the system evolves, at the next instant, into a state where the maximum number of uncontrollable events is admissible) [54],
- the *stabilization of a system* (choice of a command such that the system evolves, at the next instant, into a state with minimal change for the state variable values) [54].
- the *optimal control* (minimization of the cost of the trajectories between a set of initial states and a set of final states) [53].

5.3 Integration in the SIGNAL environment

In this section we present how the controller synthesis methodology has been integrated in the SIGNAL environment. There are two fundamental aspects: the first one deals with the unification of the formalism and the second one deals with the visualization of the result.

1. First, to simplify the use of the tool, the same language is now used to specify the physical model of the system and the control objectives (as well as the verification objectives). Both can now be written in a new extension of the SIGNAL language, named SIGNAL+. With SIGNAL+, it is not necessary for the user to know (and/or to understand) the mathematical framework utilized in the computation of controllers.

2. Moreover, some obstacles prevent the diffusion of formal methods for logical controller synthesis. The most important deals with the abstraction of the obtained controllers, coded, in our framework, by Ternary Decision Diagrams (TDDs) (see Appendix B). The result is in general too complex to be satisfactorily understood.

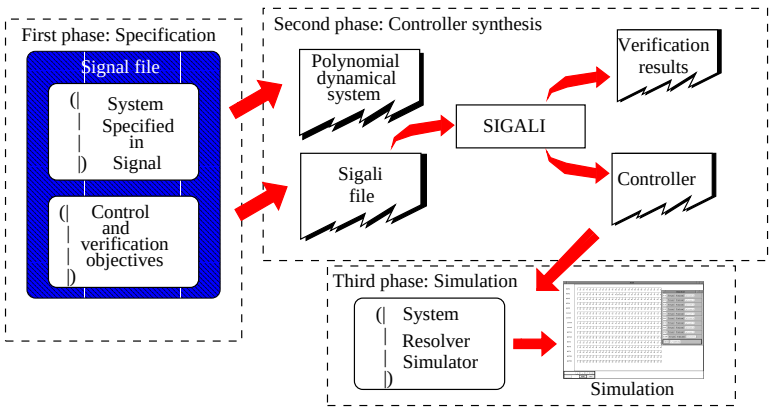


Figure 3: Description of the tool

We developed a tool allowing the controller synthesis as well as the visualization of the result by interactive simulation of the controlled system. Figure (3) sums up the different

stages necessary to perform such simulations. In the first stage, the user specifies the physical model and the control objectives in SIGNAL. The second stage is performed by the SIGNAL compiler which translates the initial SIGNAL program into a PDS and the control objectives in terms of polynomial relations and operations. The controller is then synthesized, using SIGALI. Finally, in the third stage, the obtained controller is included in the initial SIGNAL program in order to perform simulation.

5.3.1 First phase: Specification of the model

The physical model is first specified in the language SIGNAL. It describes the global behavior of the system. In the same stage we specify a process, that describes all the properties that must be enforced on the system. This process can also contain some property verification objectives. Using an extension of the SIGNAL language, SIGNAL+, it is possible to express the properties to be checked as well as the control objectives to be synthesized, directly, in the SIGNAL program. The syntax is:

```
(| Sigali(Verif_Objective(Prop))
 | Sigali(Control_Objective(Prop))
 |)
```

The keyword `Sigali` means that the subexpression has to be evaluated by SIGALI. The function `Verif_Objective` (it could be `invariance`, `reachability`, `attractivity`, among others) means that SIGALI has to check the corresponding property according to the boolean `PROP`. The function `Control_Objective` means that SIGALI has to compute a controller that ensures the control objective for the controlled system. We also precise in the SIGNAL program the status of the inputs (controllable or not)). It is done by the function **Controllable()**.

The complete SIGNAL program is obtained by composing the two processes in parallel.

```
(| System() %Physical model in Signal%
 | Objectives() %verif. and control Objectives%
 |)
```

5.3.2 Second phase: Controller Synthesis

In order to perform the computation of the controller with regard to the different control objectives, the SIGNAL compiler produces a file which contains the PDS resulting from the abstraction of the complete SIGNAL program and the algebraic control (as well as verification) objectives. We thus obtain a file that can be read by SIGALI.

Suppose that we must enforce, in a SIGNAL program named "system.SIG" the invariance of the set of states where the boolean `PROP` is *true*. The corresponding SIGNAL program is then given by:

```
(| (| system{} %the physical specified in Signal%
   |)
 | PROP : definition of the boolean PROP in Signal
 | Sigali(S_Invariance(True(PROP))
 |)
```

The corresponding SIGALI file, obtained after the compilation of the global SIGNAL program, is

```
read(‘‘system.z3z’’);  => loading of the PDS
Set_States : True(PROP)
 => Compute the  states where PROP is true
S_c: S_Invariance(S,Set_States) => Synthesize the controller that ensures
                                   the invariance of Set_States
```

The file “system.z3z” contains in a coded form the PDS that represents the system. The `PROP` signal becomes a polynomial `Set_States` expressed by state variables and events, which is equal to 0 when `PROP` is *true*. The last line of the file consists in synthesizing a controller which ensure the invariance of the set of states where the polynomial `Set_States` takes the value 0. This file is then interpreted by SIGALI that computes the controller. The result is a polynomial, encoded by a TDD which is saved in a file that can be used to perform simulation.

5.3.3 Third phase: Result Simulation

To obtain a simulation allowing the visualization of the controlled system new behavior, the controller (more precisely, a resolver process) is automatically integrated in the initial SIGNAL program as well as simulation processes following the scheme of Figure (4).

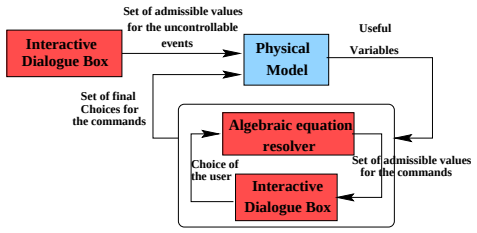


Figure 4: The resulting SIGNAL program

Integration of the resolver in a SIGNAL program & simulator building: In our framework, a controller is a polynomial coded in a TDD. In most cases, several values are possible for each command, when the system evolves into a state. Therefore, an algebraic equation resolver has been developed in SIGNAL for the control part of the **resolver process** and in

C^{++} for the algebraic equation resolver part. This process solves polynomial equations (*i.e.*, controllers) according to the internal state values and the input event values. The constraint part of the controller is given by a polynomial $C(X, Y, U) = 0$. The resolver process provides, for given values x, y, all the possible values for the command u. Note that not only one but all the alternatives of commands are proposed. This process is automatically integrated in the initial SIGNAL program, following the diagram of Figure (4). The links (*i.e.*, the connections through signals) between the process resolver and the process which specifies the system are automatically added in order to obtain the new SIGNAL program.

At the same time, the user has the option of adding in this new program some generic processes of simulation. These SIGNAL processes perform, after compilation, the automatic construction of graphical input acquisition buttons and output display windows for the signals of the interface of the programs, in an oscilloscope-like fashion (We are also able to perform real graphical animation in order to simulate the behavior of the system [51]); with regard to the commands, the graphical acquisition button processes are automatically added in the SIGNAL program when the resolver is included. We finally compile the resulting SIGNAL program that generates executable code ready for simulation.

Simulation principle: The event values are chosen by the user under the control of the resolver through an interactive dialogue box. When a choice is performed by the user, this choice is automatically sent to the algebraic resolver, which returns the set of possible values for the remaining commands. In fact, each time a new choice is made by the user, a new controller is computed, in the sense that one variable of the polynomial controller has been instantiated. New constraints can then appear on the commands which are not totally specified. During this

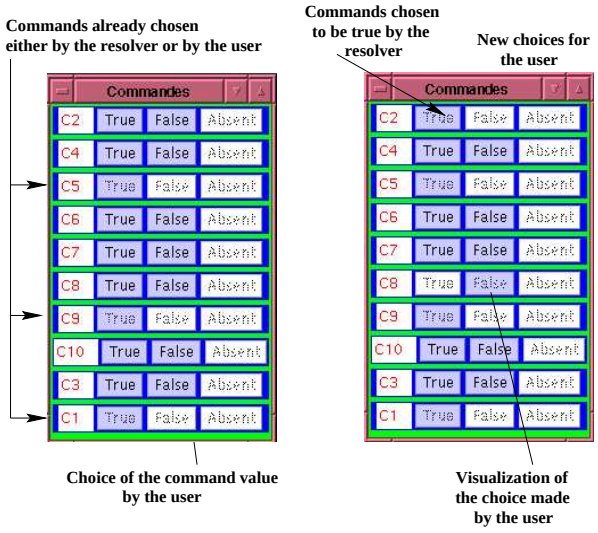


Figure 5: Example of simulation during a step.

exchange between the dialogue box and the resolver, some commands can be totally specified by the resolver in which case their values are then imposed. The choice of the command values can be performed step by step by the user, or using a random process for a step of simulation. In the second case, the resolver chooses the command values. The user can also ask for a random simulation during an indeterminate number of simulation steps.

6 Application to the incremental design of a Power Transformer Station Controller

6.1 Overview of the power transformer station

In this section, we make a brief description of the power transformer station network as well as the various requirements the controller has to handle.

6.1.1 The power transformer station description

Électricité de France has hundreds of high voltage networks linked to production and medium voltage networks connected to distribution. Each station consists of one or more power transformer stations to which circuit-breakers are connected. The purpose of an electric power transformer station is to lower the voltage so that it can be distributed in urban centers to end-users. The kind of transformer (see Figure 6) we consider, receives high voltage lines, and feeds several medium voltage lines to distribute power to end-users.

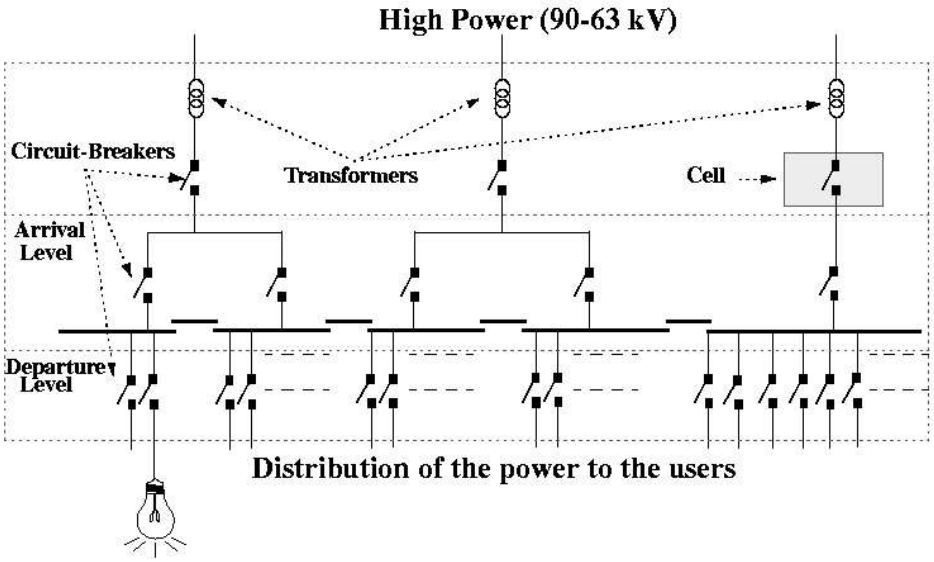
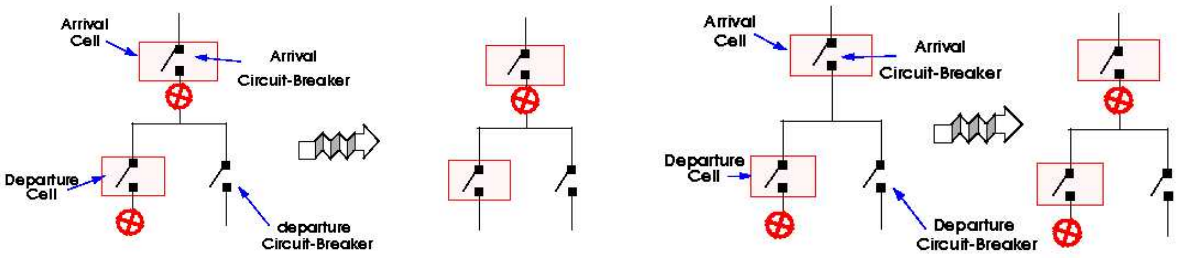


Figure 6: The power transformer station topology.

For each high voltage line, a transformer lowers the voltage. During operation of this system, several faults can occur (three types of electric faults are considered: phase `PH`, homopolar `H`, or wattmetric `W`), due to causes internal or external to the station. To protect the device and the environment, several circuit breakers are placed in a network of *cells* in different parts of the station (on the arrival lines, link lines, and departure lines). These circuit breakers are informed about the possible presence of faults by sensors.

Power and Fault Propagation: We discuss here some physical properties of the power network located inside the power transformer station controller. It is obvious that the power can be seen by the different cells if and only if all the upstream circuit-breakers are closed. Consequently, if the link circuit-breaker is opened, the power is cut and no fault can be seen by the different cells of the power transformer station. The visibility of the fault by the sensors of the cells is less obvious. In fact, we have to consider two major properties:

- On one hand, if a physical fault, considered as an input of our system, is seen by the sensors of a cell, then all the downstream sensors are not able to see some physical faults. In fact, the appearance of a fault at a certain level (the departure level in Figure 7(a) for example) increases the voltage on the downstream lines and masks all the other possible faults.



(a) The fault masking

(b) The fault propagation

Figure 7: The Fault properties

- On the other hand, if the sensors of a cell at a given level (for example the sensors of one of the departure cells as illustrated in Figure 7(b)) are informed about the presence of a fault, then all the upstream sensors (here the sensors of the arrival cell) detect the same fault. Consequently, it is the arrival cell that handle the fault.

6.1.2 The controller

The controller can be divided into two parts. The first part concerns the local controllers (*i.e.*, the cells). We chose to specify each local controller in SIGNAL, because they merge logical and numerical aspects. We give here only a brief description of the behavior of the different cells (more details can be found in [56, 42] or in [11, 10]). The other part concerns more general requirements to be checked by the global controller of the power transformer station. That specification will be described in the following.

The Cells: Each circuit breaker controller (or cell) defines a behavior beginning with the confirmation and identification of the type of the fault. In fact, a variety of faults are transient, i.e., they occur only for a very short time. Since their duration is so short that they do not cause any danger, the operation of the circuit-breaker is inhibited. The purpose of this confirmation phase is let the transient faults disappear spontaneously. If the fault is confirmed, the handling consists in opening the circuit-breaker during a given delay for a certain number of periods and then closing it again. The circuit-breaker is opened in consecutive cycles with an increased duration. At the end of each cycle, if the fault is still present, the circuit-breaker is reopened. Finally, in case the fault is still present at the end of the last cycle, the circuit-breaker is opened definitively, and control is given to the remote operator.

The specification of a large part of these local controllers has been performed using the SIGNAL synchronous language [56] and verified using our formal calculus system,SIGALI [42].

Some global requirements for the controller: Even if is quite easy to specify the local controllers in SIGNAL, some other requirements are too informal, or their behaviors are too complex to be expressed directly as programs.

1. One of the most significant problems concerns the appearance of two faults (the kind of faults is not important here) at two different departure cells, at the same time. Double faults are very dangerous, because they imply high defective currents. At the place of the fault, this results in a dangerous path voltage that can electrocute people or cause heavy material damages. The detection of these double faults must be performed as fast as possible as well as the handling of one of the faults.

2. Another important aspect is to know which of the circuit breakers must be opened. If the fault appears on the departure line, it is possible to open the circuit breaker at departure level, at link level, or at arrival level. Obviously, it is in the interest of users that the circuit be broken at the departure level, and not at a higher level, so that the fewest users are deprived of power.

3. We also have to take into account the importance of the departure circuit-breaker. Assume that some departure line, involved in a double faults problem, supplies a hospital.

Then, if the double faults occur, the controller should not open this circuit-breaker, since electricity must always delivered to a hospital.

The transformer station network as well as the cells are specified in SIGNAL. In order to take into account the requirements (1), (2) and (3), with the purpose of obtaining an optimal controller, we rely on automatic controller synthesis that is performed on the logical abstraction of the global system (network + cells).

6.2 Specification in SIGNAL of the power transformer station

The transformer station network we are considering contains four departure, two arrival and one link circuit-breakers as well as the cells that control each circuit-breaker [42]. The process `Physical_Model` in Figure 2 describes the power and fault propagation according to the state of the different circuit-breakers. It is composed of nine subprocesses. The process `Power_Propagation` describes the propagation of power according to the state of the circuit-breakers (Open/Closed). The process `Fault_Visibility` describes the fault propagation and visibility according to the other faults that are potentially present. The remaining seven processes encode the different circuit-breakers.

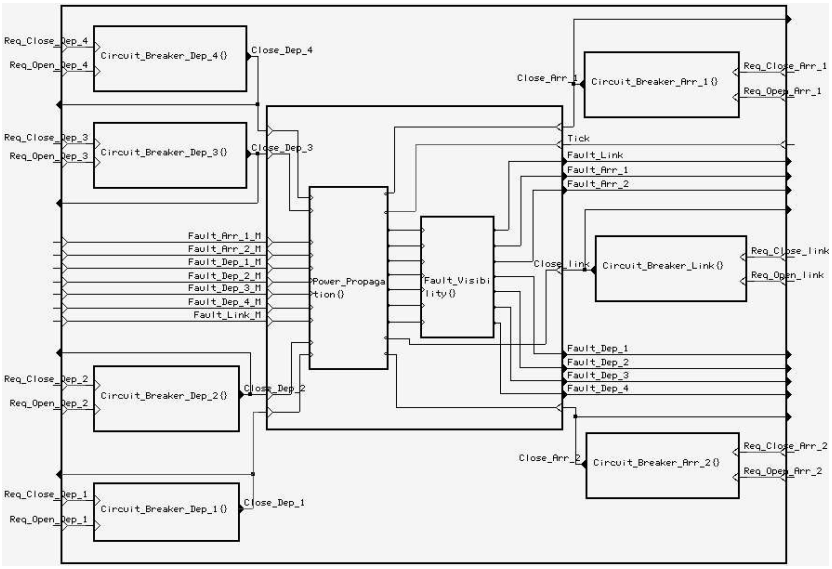


Figure 8: The main process in SIGNAL

The inputs of this main process are booleans that encode the physical faults: `Fault_Link_M`, `Fault_Arr_i_M` (`i`=1,2), `Fault_Dep_j_M` (`j` =1,..,4). They encode faults that are really present on the different lines. The event inputs `req_close_...` and `req_open_...` indicate opening and closing requests of the various circuit-breakers. The outputs of the main process are the booleans `Fault_Link`, `Fault_Arr_i`, `Fault_Dep_j`, representing the signals that are sent to

the different cells. They indicate whether a cell is faulty or not. These outputs represents the knowledge that the sensors of the different cells have.

We will now see how the subprocesses are specified in SIGNAL.

The circuit-breaker: The process `Circuit-Breaker` takes two sensors inputs: `Req_Open` and `Req_Close`. They represent opening and closing requests. The output `Close` represents the status of the circuit-breaker.

```
(| Close := (Req_Close default (false when Req_Open) default Z_Close
 | Z_Close := Close $1 init true
 | Close ^= Tick
 | (Req_Close when Req_Open) ^= when (not Req_Open)
 |)
```

Figure 9: The Circuit-breaker in SIGNAL

The boolean `Close` becomes *true* when the process receives the event `req_close`, and *false* when it receives the event `Req_open`, otherwise it is equal to its last value (i.e. `Close` is *true* when the circuit-breaker is closed and *false* otherwise). The constraint `Req_Close when Req_Open ^= when not Req_Close` says that the two events `Req_Close` and `Req_Open` are exclusive.

Power Propagation: It is a filter process using the state of the circuit-breakers. Power propagation also induces a visibility of possible faults. If a circuit-breaker is open then no fault can be detected by the sensors of downstream cells.

This is specified in the process `Power_Propagation` shown in Figure 10. The inputs are booleans that code the physical faults and the status of the circuit-breakers. For example, a fault could be detected by the sensor of the departure cell 1 (i.e. `Fault_Dep_1_E` is *true*) if there exists a physical fault (`Fault_Dep_1_M`=*true*) and if the upstream circuit-breakers are closed (ie, `Close_Link`=*true* and `Close_Arr_1`=*true* and `Close_Dep_1`=*true*).

6.2.1 Fault visibility and propagation:

The `Fault_Visibility` process in Figure 11, specifies fault visibility and propagation. As we explained in Section 6.1.1, a fault could be seen by the sensors of a cell only if no upstream fault is present.

For example, a fault cannot be detected by the sensor of the departure cell 1 (i.e. `Fault_Dep_1` is *false*), even if a physical fault exists at this level (`Fault_Dep_1_E`=*true*[15]), when another physical fault exists at the link level (i.e., `Fault_Link_1_K`=*true*) or at the arrival level 1 (i.e., `Fault_Arr_1_K`=*true*). It is thus, *true* just when the departure cell 1 detects a physical fault

[15]Note that this fault has already be filtered. It can only be present if all the upstream circuit-breakers are closed

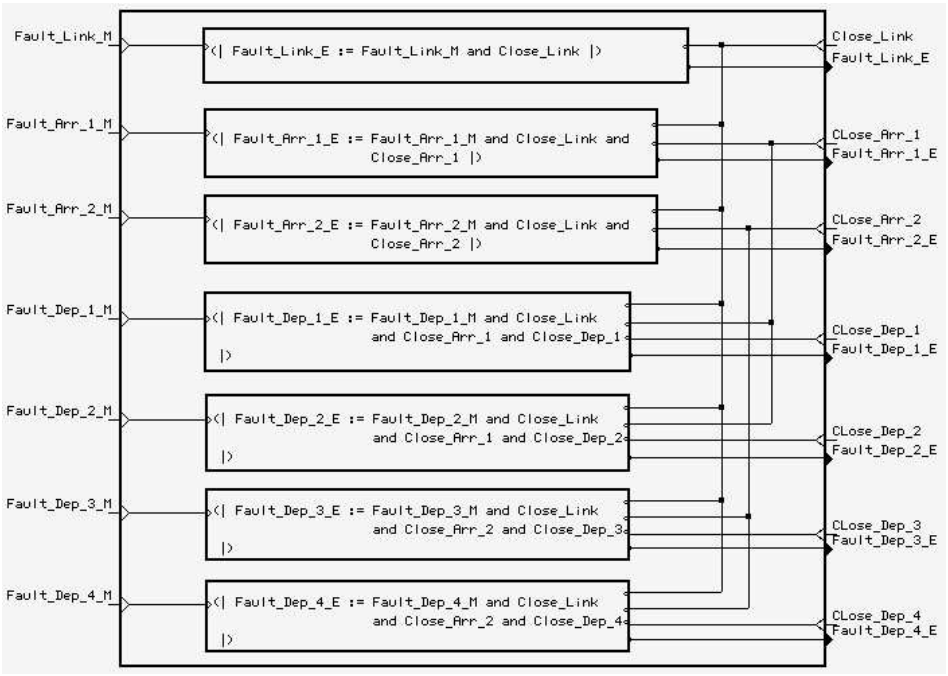


Figure 10: specification in SIGNAL of the power propagation

```
(| Fault_Link_K := Fault_Link_E
 | Fault_Arr_1_K := (not (when Fault_Link_K)) default Fault_Arr_1_E
 | Fault_Arr_2_K := (not (when Fault_Link_K)) default Fault_Arr_2_E
 | Fault_Dep_1 := (not (when Fault_Link_K)) default (not (when Fault_Arr_1_K)) default Fault_Dep_1_E
 | Fault_Dep_2 := (not (when Fault_Link_K)) default (not (when Fault_Arr_1_K)) default Fault_Dep_2_E
 | Fault_Dep_3 := (not (when Fault_Link_K)) default (not (when Fault_Arr_2_K)) default Fault_Dep_3_E
 | Fault_Dep_4 := (not (when Fault_Link_K)) default (not (when Fault_Arr_2_K)) default Fault_Dep_4_E
 | Fault_Arr_1 := (when (Fault_Dep_1 default Fault_Dep_2)) default Fault_Arr_1_K
 | Fault_Arr_2 := (when (Fault_Dep_3 default Fault_Dep_4)) default Fault_Arr_2_K
 | Fault_Link := (when (Fault_Arr_1 default Fault_Arr_2)) default Fault_Link_K
 | Fault_Link ^= Fault_Arr_2 ^= Fault_Arr_1 ^= Fault_Dep_4 ^= Fault_Dep_3 ^= Fault_Dep_2 ^=
   Fault_Dep_1 ^= Tick
 |)
```

Figure 11: Specification in SIGNAL of the fault propagation and visibility

(`Fault_Dep_1_E`) and no upstream fault exists. *A contrario*, if a fault is picked up by a cell, then it is also picked up by the upstream cells. This is for example the meaning of `Fault_Link := (when (Fault_Arr_1 default Fault_Arr_2)) default Fault_link_K`.

6.3 Verification of the power transformer network

In this section, we apply the tools to check various properties of our SIGNAL implementation of the transformer station. After the translation of the SIGNAL program, we obtain a PDS with 60 state variables and 35 event variables. Note that the compiler also checks the causal and temporal concurrency of our program and produces an executable code. We will now describe some of the different properties, which have been proved.

(1) *"There is no possibility to have a fault at the departure, arrival and link level when the link circuit-breaker is opened."* In order to check this property, we add to the original specification the following code

```
(| Error:= ((Fault_Link or Fault_Arr_1 or Fault_Arr_1 or
       Fault_Dep_1 or Fault_Dep_2 or Fault_Dep_3 or Fault_Dep_4)
            when Open_Link)  default false
 | Error ^= Tick
 | Sigali(Reachable(True(Error)))
 |)
```

The `Error` signal is a boolean which takes the value *true* when the property is violated. In order to prove the property, we have to check that there does not exist any trajectory of the system which leads to the states where the `Error` signal is *true* (`Reachable(True(Error))`). The produced file is interpreted by SIGALI that checks whether this set of states is reachable or not. In this case, the result is *false*, which means that the boolean `Error` never takes the value *true*. The property is satisfied[16] . In the same way, we proved similar properties when one of the arrival or departure circuit-breakers is open.

(2) *"If there exists a physical fault at the link level and if this fault is picked up by its sensor then the arrival sensors can not detect a fault"*. We show here the property for the arrival cell 1. It can be expressed as an invariance of a set of states.

```
(| Error:= (Fault_Arr_1 when Fault_Link_E) default false
 | Error ^= Tick
 | Sigali(Invariance(False(Error)))
 |)
```

[16] Alternatively, this property could be also expressed as the invariance of the boolean `False(Error)`, namely `Sigali(Invariance(False(Error)))`.

We have proved similar properties for a departure fault as well as when a physical fault appears at the arrival level and at the departure level at the same time.

(3) We also proved using the same methods the following property: *"If a fault occurs at a departure level, then it is automatically seen by the upstream sensors when no other fault exists at a higher level."*

All the important properties of the transformer station network have been proved in this way. Note that the cell behaviors have also been proved (see [42] for more details).

6.4 The incremental specification of the controller

6.4.1 Logical Control Problems

We have seen in the previous section, that one of the most critical requirements concerns the double fault problem. We assume here that the circuit-breakers are ideal, i.e. they immediately react to actuators (*i.e.*, when a circuit-breaker receives an opening/closing request, then at the next instant the circuit-breaker is opened/closed). With this assumption, the double fault problem can be rephrased as follows:

"if two faults are picked up at the same time by two different departure cells, then at the next instant, one of the two faults (or both) must disappear."

In order to synthesize the controller, we assume that the only controllable events are the opening and closing requests of the different circuit-breakers. The other events concern the appearance of the faults and cannot be considered controllable. The specification of the control objective is then:

```
(| 2_Fault :=         when (Fault_Dep_1 and Fault_Dep_2)
             default  when (Fault_Dep_1 and Fault_Dep_3)
             default  when (Fault_Dep_1 and Fault_Dep_4)
             default  when (Fault_Dep_2 and Fault_Dep_3)
             default  when (Fault_Dep_2 and Fault_Dep_4)
             default  when (Fault_Dep_3 and Fault_Dep_4) default false
 | Z_2_Fault := 2_Fault $1 init false
 | Error := 2_Fault and Z_2_Fault
 | Sigali(S_Invariance(S,False(Error))
 |)
```

The boolean `2_Fault` is *true*, when two faults are present at the same time and is *false* otherwise. The boolean `Error` is *true* when two faults are present at two consecutive instants. We then ask SIGALI to compute a controller that forces the boolean `Error` to be always *false* (i.e., whatever the behavior, there is no possibility for the controlled system to reach a state where `Error` is *true*).

The SIGNAL compiler translates the SIGNAL program into a PDS, and the control objectives in terms of polynomial relations and polynomial operations. Applying the algorithm, described

by the fixed-point computation (25), we are able to synthesize a controller (C_1, C_0), that ensures the invariance of the set of states where the boolean *Error* is *true*, for the controlled system $S_{C_1} = S + (C_1, C_0)$. The result is a controller coded by a polynomial and a BDD.

Using the controller synthesis methodology, we solved the double fault problem. However, some requirements have not been taken into account (importance of the lines, of the circuit-breakers,...). This kind of requirements cannot be solved using traditional control objectives such as invariance, reachability or attractivity. In the next section, we will handle this kind of requirements, using control objectives expressed as order relations.

6.4.2 Optimal Controller

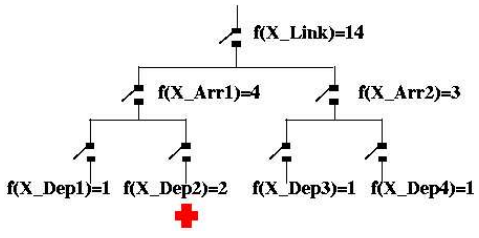
We have seen in Section 2 how to compute a controller that solves the double fault problem. However, even if this particular problem is solved, other requirements had not been taken into account. The first one is induced by the obtained controller itself. Indeed, several solutions are available at each instant. For example, when two faults appear at a given instant, the controller can choose to open all the circuit-breakers, or at least the link circuit-breaker. This kind of solutions is not admissible and must not be considered. The second requirements concerns the importance of the lines. The first controller (C_1, C_0) does not handle this kind of problems and can force the system to open the bad circuit-breakers.

As consequences, two new requirements must be added in order to obtain a real controller:

1. The number of opened circuit-breaker must be minimal
2. The importance of the lines (and of the circuit-breakers) has to be different.

These two requirements introduce a quantitative aspect to the control objectives. We will now describe the solutions we proposed to cope with these problems.

First, let us assume that the state of a circuit-breaker is coded with a state variable according to the following convention: the state variable i is equal to 1 if and only if the corresponding circuit-breaker i is closed. CB is then a vector of state variables which collects all the state variables encoding the states of the circuit-breakers. To minimize the number of open circuit-breaker and to take into account the importance of the line, we use a cost function . We simply encode the fact that the more important is the circuit-breaker, the larger is the cost allocated to the state variable which encodes the circuit-breaker. The following picture summarizes the way we allocate the cost.



The cost allocated to each state variable corresponds to the cost when the corresponding circuit-breaker is opened. When it is closed, the cost is equal to 0. The cost of a global state is simply obtained by adding all the circuit-breaker costs. With this cost function, it is always more expensive to open a circuit-breaker at a certain level than to open all the downstream circuit-breakers. Moreover, the cost allocated to the state variable that encodes the second departure circuit-breaker (encoded by the state variable X_{dep2})) is bigger than the others because the corresponding line supplies a hospital (for example). Finally note that the cost function is minimal when the number of open circuit-breaker is minimal.

Let us consider the system S_{C_1}. We then introduce an order relation over the states of the system: a state x_1 is said to be better compared to a state x_2 ($x_1 \sqsupseteq x_2$) if and only if for their corresponding sub-vectors CB_1 and CB_2, we have $CB_1 \sqsupseteq_c CB_2$. This order relation is then translated in an algebraic relation $R_{\sqsupseteq_c}$, following Equation (34) and by applying the construction described in proposition 11 and 8, we obtain a controller (C_2, C'_0) for which the controlled system $S_{C_2} = (S_{C_1} + (C_2, C'_0))$ respects the control strategy.

7 Conclusion

In this article, we have shown the feasibility of controller synthesis for the class of polynomial dynamical systems over $\mathbb{Z}/3\mathbb{Z}$. All the concepts rely on a small kernel of simple operations (see Appendix A).

In this framework, systems of polynomial equations characterize a set of solutions that encode states and events. The techniques used in this framework consist in manipulating the equation systems instead of the solution sets, and thus avoiding the enumeration of the state space. All the computations are then performed *symbolically.* Operations used on equation systems belong to the theory of elementary algebraic geometry, such as varieties, ideals and comorphisms. They enable the treatment of properties such as *safety, reachability,* and *attractivity.* Using the same framework, controller synthesis problem can be efficiently solved. Beyond synthesis of classical control objectives, we showed that the same algebraic framework also applies to handle order relations. These relations can be used to synthesize control objectives talking about the manner to reach a logical goal.

The Polynomial Dynamical System model results from the translation of a SIGNAL program [43]. We have a powerful environment to describe the model for real-time data-flow system. Moreover, the formal specification, in terms of natural language of a plant is in general, hybrid, in the sense that, it contains a purely logic part and a numerical one. This aspect can be easily handled by SIGNAL. The fact that the SIGNAL compiler automatically performs the logic/numeric separation is one of the main advantage of this technique. It allows us to directly applied the controller synthesis techniques on the logical part of the plant by abstracting away the numerical part without having to bother with this separation during the specification of the plant. Finally, both property verification and controller synthesis algorithms have been implemented in the formal proof system SIGALI for the case of polynomial dynamical system

over $\mathbb{Z}/3\mathbb{Z}$. However all these techniques and algorithms can be extended to any $\mathbb{Z}/p\mathbb{Z}$, with p prime and, in particular, in $\mathbb{Z}/2\mathbb{Z}$, where BDD packages can be efficiently used to implement polynomial operations as done for example by [31, 59]. These methods have finally been applied to the incremental design of a power transformer station controller [42, 54, 57].

The theory of polynomial dynamical systems over $\mathbb{Z}/3\mathbb{Z}$ (or $\mathbb{Z}/p\mathbb{Z}$) deserves much more research. One issue is the optimal control under partial observations, using state aggregation techniques as carried out in [46] or in a slightly different domain the control of implicit non-deterministic polynomial dynamical systems. Some other perspectives concern the synthesis of fault tolerance controllers like in [67], the synthesis of controllers using bisimulation techniques [8, 7, 47].

References

[1] T. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten. Signal – the specification of a generic, verified production cell controller. *Formal Development of Reactive Systems – Case Study Production Cell, LNCS 891, Chp. VII*, (891):115–129, January 1995.

[2] A. Arnold. *Finite transition systems*. Prentice Hall, NJ, 1994.

[3] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. *Lecture Notes in Computer Science*, 999:1–??, 1995.

[4] M.F. Atiyah and I.G. MacDonald. *Introduction to commutative Algebra*. Addison-Welsey, 1969.

[5] B. Alpern and F.B. Schneider. Recognizing Safety and Liveness. *Distributed Comp.*, 2:117–126, 1987.

[6] S. Balemi, G. J. Hoffmann, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, July 1993.

[7] G. Barret and S. Lafortune. Bisimulation, the supervisory control problem and strong model matching for finite state machines. *Journal of Discrete Event Dynamic Systems*, 8(4):337–429, December 1998.

[8] G. Barrett and Lafortune S. Using bisimulation to solve discrete event control problems. In *Proc. 1997 American Control Conf.*, pages 2337–2341, Albuquerque, NM, June 1997.

[9] A. Benveniste and P. Le Guernic. Hybrid dynamical systems and the signal programming language. *IEEE Trans. Automat. Control*, 35:535–546, May 1990.

[10] N. Bergé. *Modélisation au moyen des réseaux de Petri temporisés stochastiques d'une application de contrôle-commande de poste de transformation d'énergie électrique répartie sur le réseau de terrain FIP.* PhD thesis, Université paul Sabatier de Toulouse, 1996.

[11] N. Bergé, R. jacinto, and Samaan M. design of a transformer station control system distributed on the fielbus fip. In *Symp. on Control of power Plants and Power Systems (SIPOWER'95)*, December 1995.

[12] G. Berry. Real time programming: Special purpose languages or general purpose languages. In *11th IFIP World Congress*, pages 11–17, San Francisco, Californie, August 1989.

[13] P. Bournai and P. Le Guernic. Un environnement graphique pour le langage signal. Technical Report 741, IRISA, September 1993.

[14] F. Boussinot and R. de Simone. The ESTEREL language. *Proceedings of the IEEE*, 9(79):1293–1304, September 1991.

[15] R.E. Bryant. Graph-based algorithms for boolean function manipulations. *IEEE Transaction on Computers*, C-45(8):677–691, August 1986.

[16] R.E. Bryant and Chen Y. Verification of Arithmetic Functions with Binary Diagrams. Research Report, School of Computer Science CMU, May 1995.

[17] B. Buchberger. A theorical basis for the reduction of polynomials to canonical form. In *ACM SIGSAM bull.*, volume 10(3), pages 19–29, 1976.

[18] B. Buchberger. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232, Dordrecht, 1985. K. Reidel Publishing Company.

[19] J. Chazarain, L. A. Alonso, and E. Briales. Multivalued Logic and Gröbner bases with Applications to Modal Logic. In *Journal of Symbolic Computation*, volume 11, 181-194 1991.

[20] O Coudert and J.-C. Madre. A verified framework for the formal verification of sequential circuit. In *Conference on Computer- Aided design, IEEE*, pages 126–130, November 1990.

[21] D. Cox, J. Little, and D. O'Shea. *Ideals, varieties and algorithms : an introduction to computational algebraic geometry and commutative algebra.* Springer-Verlag, 1993.

[22] B. Dutertre. *Spécification et preuve de systèmes dynamiques.* PhD thesis, Université de Rennes I, IFSIC, December 1992.

[23] B. Dutertre and M. Le Borgne. Control of polynomial dynamic systems: an example. Research Report 798, IRISA, January 1994.

[24] T. Gautier and P. Le Guernic. Code generation in the sacres project. In *Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99*, Huntingdon, UK, February 1999. Springer.

[25] R. Germundsson. Basic results on ideals and varieties in finite fields. Technical Report LiTH-ISY-I-1259, Linköping University, September 1991.

[26] R. Germundsson. *Symbolic Systems - Theory, Computation and Applications*. PhD thesis, Linköping University, 1995.

[27] J. Gunnarsson. *Symbolic Methods and Tools for Discrete Event Dynamic Systems*. PhD thesis, Linköping University, 1997.

[28] J. Gunnarsson and Plantin J. Automatic synthesis for simultaneus supervision and control - a first example. Technical Report LiTH-ISY-R-, Linköping University, 1996.

[29] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer, 1993.

[30] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[31] G. Hoffmann and H. Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proc. of 1992 American control Conference, Chicago,Il,USA*, pages 2789–2793, 1992.

[32] L. E. Holloway and B. H. Krogh. Controlled Petri nets: A tutorial survey. In G. Cohen and J. P. Quadrat, editors, *11th Int. Conf. on Analysis and Optimization of Systems, Discrete Event Systems*, volume 199 of *LNCS*, pages 158–168, Berlin, Germany, June 1994. Springer-Verlag.

[33] L.E. Holloway, B.H. Krogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Application*, 7:151–190, 1997.

[34] A Ichikawa and K. Iraishi. Analysis and control of discrete-events sytems represented by petri nets. In *Discrete Event Sytems: Models and Applications, IIASA conf.*, Sopron, Hungary, August 1987.

[35] O. Kouchnarenko and S. Pinchinat. Intensional approachs for symbolic methods. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[36] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[37] R. Kumar and V. K. Garg. Optimal control of discrete event dynamical systems using network flow techniques. In *Proc. of 29th Allerton Conf. on Communication, Control and Computing*, Champaign, IL, USA, October 1991.

[38] M. Le Borgne. *Systèmes dynamiques sur des corps finis*. PhD thesis, Université de Rennes I, September 1993.

[39] M. Le Borgne, A. Benveniste, and P. Le Guernic. Polynomial ideal theoric methods in discrete event and hybrid dynamical systems. In *Proc. of 28^{t}h IEEE Conf. on Decision and Control*, pages 2695–2700, Tampa, FL, December 1989.

[40] M. Le Borgne, A. Benveniste, and P. Le Guernic. Polynomial dynamical systems over finite fields. In *Algebraic Computing in Control*, volume 165, pages 212–222. LNCIS, G. Jacob et F. Lamnabhi-lagarrigue, March 1991.

[41] M. Le Borgne, A. Benveniste, and P. Le Guernic. Dynamical systems over Galois fields and DEDS control problems. In *Proc. of the 30th IEEE conference on Decision and Control*, pages 1505–1510, 1992.

[42] M. Le Borgne, H. Marchand, E. Rutten, and M. Samaan. Formal verification of signal programs: Application to a power transformer station controller. In *Proceedings of AMAST'96*, pages 271–285, Munich, Germany, July 1996. Springer-Verlag, LNCS 1101.

[43] P. Le Guernic and T. Gautier. Data-flow to von Neumann: the SIGNAL approach. In Jean-Luc Gaudiot and Lubomir Bic, editors, *Advanced Topics in Data-Flow Computing*, chapter 15, pages 413–438. Prentice-Hall, 1991.

[44] Y. Li and W.M. Wohnam. Control of discrete-event systems-part i : the base model. *IEEE Trans. Automatic Control*, 38(8):1214–1227, August 1993.

[45] Y. Li and W.M. Wohnam. Control of discrete-event systems-part ii : controller synthesis. *IEEE Trans. Automatic Control*, 39(3):512–531, March 1994.

[46] Y. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[47] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. Technical Report TCS-98-2, SPIC Mathematical Institute, 1998.

[48] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E. W. Mayr and C. Puech, editors, *Proceedings STACS'95*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer-Verlag, 1995.

[49] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *12th Annual Symposium on Theoretical Aspects of Computer Science*, volume 900 of *lncs*, pages 229–242, Munich, Germany, 2–4 March 1995. Springer.

[50] H Marchand, O. Boivineau, and Lafortune S. On the synthesis of optimal schedulers in discrete event control problems with multiple goals. In *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, pages 734–739, San Diego, California, USA, October 1998.

[51] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. A design environment for discrete-event controllers based on the signal language. In *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, pages 770–775, San Diego, California, USA, October 1998.

[52] H. Marchand and M. Le Borgne. Partial order control and optimal control of discrete event systems modeled as polynomial dynamical systems over galois fields. Technical Report 1125, IRISA, October 1997.

[53] H. Marchand and M. Le Borgne. On the optimal control of polynomial dynamical systems over $\mathbb{Z}/_{p\mathbb{Z}}$. In *4th International Workshop on Discrete Event Systems*, pages 385–390, Cagliari, Italy, August 1998.

[54] H. Marchand and M. Le Borgne. Partial order control of discrete event systems modeled as polynomial dynamical systems. In *1998 IEEE International Conference On Control Applications*, Trieste, Italia, September 1998.

[55] H. Marchand, E. Rutten, and M. Samaan. Specifying and verifying a transformer station in SIGNAL and SIGNAL*GTi*. Research Report 2521, INRIA, March 1995.

[56] H. Marchand, E. Rutten, and M. Samaan. Synchronous design of a transformer station controller with Signal. In *4th IEEE Conference on Control Applications*, pages 754–759, Albany, New-York, September 1995.

[57] H. Marchand and M. Samaan. Incremental design of a power transformer station controller using controller synthesis methodology. In *World Congress on Formal Methods (FM'99)*, pages 1605–1624, Toulouse, France, September 1999. Springer Verlag, LNCS 1709.

[58] H. Matsumura. *Commutative Ring Theory*. Cambridge Studies in advanced Mathematics 8. Cambridge University Press, 1986.

[59] H. Melcher and K. Winkelmann. Controller synthesis for the production cell case study. In *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP-98)*, pages 24–33, New YOrk, March 4–5 1998. ACM Press.

[60] D. Mumford. The red book of varieties and schemes. *Lecture Notes in Math. 1388, Springer*, 1988.

[61] K. M. Passino and P. J. Antsaklis. On the optimal control of discrete event systems. In *Proc. of 28th Conf. Decision and Control*, pages 2713–2718, Tampa, Floride, December 1989.

[62] S. Pinchinat. Sigali *vs* μcalculus. Personnal communication, 1996.

[63] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM J. Control Optim.*, 25(5):1202–1218, September 1987.

[64] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[65] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.

[66] M. Reid. *Undergraduate Algebraic Geometry*, volume 12. Cambridge University Press, london mathematical society student texts edition, 1988.

[67] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. In *Proc of the 11th Int. Conf. on Analysis and Optimization of Systems, D.E.S.*, LNCS No 953, pages 73–79, June 1994.

[68] R. Sengupta and S. Lafortune. Extensions to the theory of optimal control of discrete event systems. In S. Balemi, P. Kozák, and R. Smedinga, editors, *Discrete Event Systems: Modeling and Control*, volume 13 of *Progress in Systems and Control Theory*, pages 153–160. Birkhäuser Verlag , Basel, Switzerland, 1993. (Proc. of WODES'92, Aout 1992, Prague, Czechoslovakia).

[69] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM Journal on Control and Optimization*, 36(2), March 1998.

[70] I.R. Shafarevich. *Basic Algebraic Geometry*. Springer, 1974.

[71] J. C. Willems. Paradigms and Puzzles in the Theory of Dynamical Systems. *IEEE Transactions on Automatic Control*, 36(3):258–294, 1991.

A The algebraic Tool box

In the preceding sections, a set of algorithms was presented to solve various problems for PDS. In fact, it turns out that all the introduced algorithms use only a small kernel of primitive operations involving polynomials and ideals. We summarize, in the following list, the basic toolkit of primitive algorithms that are the building blocks for more complex computations:

- Check if a polynomial q belongs to an ideal $\underline{a}$, and, more generally, if an ideal $\underline{a}$ is included in another ideal $\underline{b}$.
- Compute the ideal associated with the complement of a set.
- Compute intersections of the form $\underline{a} \cap A[X]$ for some ideal $\underline{a}$ in $A[X,Y]$. This allows us to compute $\exists elim_Y()$.
- Compute the $\forall elim_Y$ of some ideal.
- Compute $P^*(\underline{a})$, the image of ideal $\underline{a}$ by comorphism P^*. This is the basic tool for analyzing state transitions.

The actual implementations of these primitive algorithms are partially based on the fact that an ideal can be represented by a single polynomial, called principal generator.

A.1 Principal generator of an ideal

The previous transformations result in relations between ideals which can be verified using formal calculus. We choose to represent an ideal $\underline{a}$ by a single generator, called the *principal generator* of $\underline{a}$[17].

Proposition 16 *let $\underline{a}$ be an ideal of $A[Z]$ and $\{g_1, \ldots, g_k\}$ a set of generators of $\underline{a}$. Then the polynomial function*

$$f \quad = \quad 1 - \prod_{i=1}^{k} (1 - g_i^2) \tag{38}$$

is a principal generator of $\underline{a}$ meaning that $\underline{a} =< f >$.

Proof: Let $\underline{b}$ be the ideal generated by f. The development of $\prod_{i=1}^{k}(1-g_i^2)$ yields an expression of the form $1 + h$ where h is a combination of the polynomial functions $g_1, \ldots, g_k$. h is then

[17]Note that in [26] or in [28, 27], they chose to encode their ideals by a Gröbner basis [17, 18]. However the computation of the canonical generators of such a base can be very expensive even for a low number of variables (< 20).

an element of $\underline{a}$. From Relation (38), $f = 1 - (1 + h) = -h$ also belongs to $\underline{a}$ and then $\underline{b} \subseteq \underline{a}$. Reciprocally, for any g_i, we have

$$fg_i = g_i - g_i \prod_{j=1}^{k}(1 - g_j^2) = g_i - (g_i - g_i^3) \prod_{j=1, j\neq i}^{k} (1 - g_j^2).$$

In $A[Z]$, $g_i - g_i^3 = 0$ then $fg_i = g_i$. This shows that $g_i \in \underline{b}$ and finally $\underline{a} \subseteq \underline{b}$. ◇

Another way to obtain the principal generator of an ideal is to compute the expression:

$$f = g_1 \oplus g_1 \oplus \ldots \oplus g_k \tag{39}$$

where the operator $\oplus$ is defined by: $f \oplus f' \stackrel{def}{=} (f^2 + f'^2)^2$. We can easily show that (39) and (38) are equal. This particular generator will be used in the practical implementation of the algorithms on ideals. Now, knowing a principal generator of $\underline{a}$, it is easy to check if a polynomial f belongs to $\underline{a}$.

Proposition 17 *A polynomial $f \in \underline{a} =< g >$ in $A[Z]$ if and only if $f(1 - g^2) = 0$.*

Proof: If $f(1 - g^2) = 0$ then $f = fg^2$ and f belongs to $\underline{a}$. Reciprocally, if $f \in \underline{a}$, there exists a polynomial h such that $f = gh$ and then $f(1 - g^2) = h(g - g^3) = 0$ in $A[Z]$. ◇

The link between principal generator and varieties is:

Proposition 18 *Let E be a subset of $(\mathbb{Z}/3\mathbb{Z})^l$. A polynomial g of $A[Z]$ is a principal generator of $\mathcal{I}(E)$ if and only if, for any element $z \in (\mathbb{Z}/3\mathbb{Z})^l$: $z \in E \Rightarrow g(z) = 0$ and $z \notin E \Rightarrow g(z) \neq 0$*

Proof: Let g be a polynomial such that $\forall z \in E,\ g(z) = 0$ and $\forall z \notin E,\ g(z) \neq 0$. Then the ideal generated by g is included in $\mathcal{I}(E)$. Reciprocally, consider a polynomial $f \in \mathcal{I}(E)$. We have for all $z \in E,\ f(z) = 0$ and for all $z \notin E,\ g(z) \neq 0$. Then, for all $z \in (\mathbb{Z}/3\mathbb{Z})^l,\ f(z)(1-g^2(z)) = 0$. From Proposition 17, f belongs to the ideal generated by g.

Reciprocally, let g be a polynomial of $\mathcal{I}(E)$. By definition, g satisfies the condition $\forall z \in E,\ g(z) = 0$. If E is equal to $(\mathbb{Z}/3\mathbb{Z})^l$, then the second condition is satisfied. Otherwise, let $z = (z_1, \ldots, z_l) \in (\mathbb{Z}/3\mathbb{Z})^l - E$. We construct the polynomial P_z:

$$p_z = \prod_{i=1}^{l}(1 - (Z_i - z_i)^2).$$

We have $P_z(z) = 1$ and $P_z(z') = 0$ for all $z' \neq z$ and in particular for all $z' \in E$. Then $P_z \in \mathcal{I}(E)$ and from Proposition 17, $P_z(1 - g^2) = 0$. In other words, for all $z' \in (\mathbb{Z}/3\mathbb{Z})^l$, $P_z(z')(1 - g^2(z')) = 0$. For all $z' = z$, we have $1 - g^2(z) = 0,\ g(z)$ is not equal to null. ◇

A.2 Basic Operations

By coding ideals with a single polynomial, the ideal computations are reduced to simple polynomial operations. Hence, instead of enumerating the elements of sets and manipulating them explicitly, this approach manipulates the polynomial functions characterizing their set. The basic operations between ideals (and varieties) are summarized in the following proposition:

Proposition 19 *Let V_1 and V_2 be two varieties of $(\mathbb{Z}/_{3\mathbb{Z}})^l$ and g_1 (resp. g_2) be the principal generator of $\mathcal{I}(V_1)$ (resp. $\mathcal{I}(V_2)$) of $A[Z]$, then*

1. $V_1 \subseteq V_2 \Leftrightarrow \mathcal{I}(V_1) \supseteq \mathcal{I}(V_2) \Leftrightarrow g_2(1-g_1^2) = 0$
2. *$g_1 \oplus g_2$ is the principal generator of $\mathcal{I}(V_1) + \mathcal{I}(V_2)$*
3. *$g_1^2 g_2^2$ is the principal generator of $\mathcal{I}(V_1 \bigcup V_2)$*
4. *$1 - g_1^2$ is the principal generator of $\mathcal{I}((\mathbb{Z}/_{3\mathbb{Z}})^l - V_1)$*

Proof:

- As $\mathcal{I}(V_1) \supseteq \mathcal{I}(V_2)$, we can deduce from Proposition 17, that $g_2(1-g_1^2) = 0$. Reciprocally, assume that $g_2(1-g_1^2) = 0$. Consider $x \in V_1$ then $g_1(x) = 0$ and $g_2(x)(1-g_1^2(x)) = g_2(x) = 0$ and finally, $x \in V_2$
- From Remark 1, $\mathcal{I}(V_1) + \mathcal{I}(V_2)$ is an ideal generated by the base $\{g_1, g_2\}$ and finally a principal generator of this ideal is $g_1 \oplus g_2$ (Proposition 16).
- Let $\underline{a}$ be the polynomial generated by the polynomial $g_1^2 g_2^2$, then we have $g_1^2 g_2^2 (1-g_1^2) = g_1^2 g_2^2 - g_1^2 g_2^2 = 0$ (resp $g_1^2 g_2^2 (1-g_2^2) = 0$). So, from Proposition 17, $\underline{a} \subseteq \mathcal{I}(V_1)$ (resp. $\underline{a} \subseteq \mathcal{I}(V_2)$). The converse is similar.
- The proof is a direct consequence of Proposition 18. ◇

$\exists elim_Y()$ and $\forall elim_Y()$: Let us consider a variety E of $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ and the associated ideal $\mathcal{I}(E) \in A[X,Y]$. We just give here the sketch for the $\forall elim_Y()$ computation. In the following, we will note U_i the following variety:

$$U_i = \{(x_1, .., x_n, y_{i+1}, .., y_m) / \forall (y_1, .., y_i) \ s.t. \ (x_1, .., y_m) \in E\}$$

Using Relation (10) and Proposition 4 is easy to show that $U_m = \mathcal{V}(\forall elim_Y(\mathcal{I}(E))$. The computation of the principal generator of $\forall elim_Y(\mathcal{I}(E)$ is obtained as follows: Let g_E be the principal generator of the ideal $\mathcal{I}(E)$, we note g_1, g_2 and g_3 the three following polynomials:

$$\begin{cases} g_1 &= g_E(X_1, \ldots, X_n, 1, Y_2, \ldots, Y_m), \\ g_2 &= g_E(X_1, \ldots, X_n, -1, Y_2, \ldots, Y_m), \\ g_3 &= g_E(X_1, \ldots, X_n, 0, Y_2, \ldots, Y_m). \end{cases}$$

It is then easy to prove that, using polynomials g_1, g_2 and g_3, the polynomial g_E can be rewritten as follows:

$$g = (-Y_1 - Y_1^2)g_1 + (Y_1 - Y_1^2)g_2 + (1 - Y_1^2)g_3 \tag{40}$$

Using this decomposition, we have:

Proposition 20 *With the previous notations, $g_1 \oplus g_2 \oplus g_3$ is the principal generator of the ideal $\mathcal{I}(U_1)$.*

Proof: An element $(x_1, \ldots, x_n, y_2, \ldots, y_m)$ belongs to U_1 if and only if $\forall y_1 \in \mathbb{Z}/_{3\mathbb{Z}}$, s.t. $g_E(x_1, \ldots, y_m) = 0$; in other words if and only if all the three polynomial functions g_1, g_2 and g_3 are null. Let h be a polynomial such that $h = g_1 \oplus g_2 \oplus g_3$, U_1 is then the set of $(x_1, .., x_n, y_2, .., y_m)$ s.t. $h(x_1, .., x_n, y_2, .., y_m) = 0$. hence the result. ◇

The principal generator of the ideal $\forall elim_Y(\mathcal{I}(E))$ can be computed from the principal generator of the ideal $\mathcal{I}(E)$ associated to the variety E by repeated application of the former proposition on the set of variables Y.

If g_E is the principal generator of $\mathcal{I}(E)$, we compute the following sequence of polynomials $(u_i)_{1 \leq i \leq m}$ defined by:

$$\begin{cases} u_0 & = & g_E \\ u_{i+1} & = & u_1|_{y_i=1} \oplus u_1|_{y_i=-1} \oplus u_1|_{y_i=0} \end{cases} \tag{41}$$

For all i, $1 \leq i \leq m$, u_i is a principal generator of the ideal $\mathcal{I}(U_i)$. The polynomial u_m is then the principal generator of the ideal $\forall elim_Y(\mathcal{I}(E))$.

Comorphism $P^*()$: it is reduced to elementary polynomial function operations (sums and products) on polynomials.

A.3 Verification and controller synthesis algorithms

With the previous notations, basic geometric properties of the PDSs can be rewritten as follows: Let us consider a PDS S and a set of states E represented by its principal generator g_E, then

1. S is live iff $P^*(\exists elim_Y(Q))(1 - Q^2) = 0$,
2. E is invariant iff $P^*(g_E)(1 - Q \oplus g_E)) = 0$,
3. E is invariant under control if and only if $\exists elim_Y(Q \oplus P^*(g_E))(1 - g_E^2) = 0$.

The computation of the largest invariant (resp. invariant under control, U-invariant under control) uses the three operators pre, $\widetilde{pre}$ and $\widetilde{\widetilde{pre}}$. If E is a set of states, the canonical

generators of $pre(E)$, $\widetilde{pre}\ (E)$ and $\widetilde{\widetilde{pre}}\ (E)$ are respectively:

$$g_{pre(E)} = \exists elim_Y(P^*(g_E) \oplus Q) \quad (42)$$

$$g_{\widetilde{pre}(E)} = \forall elim_Y(P^*(g_E) \times (1 - Q^2)) \quad (43)$$

$$g_{\widetilde{\widetilde{pre}}(E)} = \forall elim_Y((1 - \exists elim_U(Q))\exists elim_U(Q \oplus P^*(g_E))) \quad (44)$$

Finally, the computation of the largest invariant, invariant under control and U-invariant under control are obtained as a fix-point computation on the corresponding principal generators.

B Polynomial Representations

To implement the operations on principal generators, a symbolic calculus system SIGALI is available. Using decomposition of a polynomial (relation (40)), it is possible to represent polynomial functions by Ternary Decision Diagrams (TDD) [40], a slight extension of Binary Decisions Diagrams (BDDs) [15], which are very efficient in boolean algebra and other areas ([20] for boolean circuits verification), are used to implement polynomials with all standard operations. Like for the BDD, we obtain in this way a canonical representation of the polynomial functions.

In the quotient ring $A[X] = \mathbb{Z}/_{3\mathbb{Z}}[X]/_{<X^3-X>}$, for each variable X_i, let us define the three polynomial functions: $e_i^1 = -X_i^2 - X_i$, $e_i^2 = -X_i^2 + X_i$, $e_i^3 = 1 - X_i^2$. These polynomial functions have interesting properties: $(e_i^\alpha)^2 = e_i^\alpha$ for $\alpha = 1, 2, 3$, $e_i^\alpha e_i^\beta = 0$ for all $\alpha \neq \beta$, and $e_i^1 + e_i^2 + e_i^3 = 1$ (In $\mathbb{Z}/_{p\mathbb{Z}}$, this family of polynomials correspond to the Lagrange polynomials).

Proposition 21 *Each $P(X) \in A[X]$ can be decomposed in a unique way as $P(X) = e_1^1 P_1 + e_2^1 P_2 + e_1^3 P_3$, where polynomials P_1, P_2, P_3 have the following form :*

$$P_1 = P(1, X_2, \ldots, X_n),\ P_2 = P(-1, X_1, \ldots, X_n),\ P_3 = P(0, X_1, \ldots, X_n,).$$

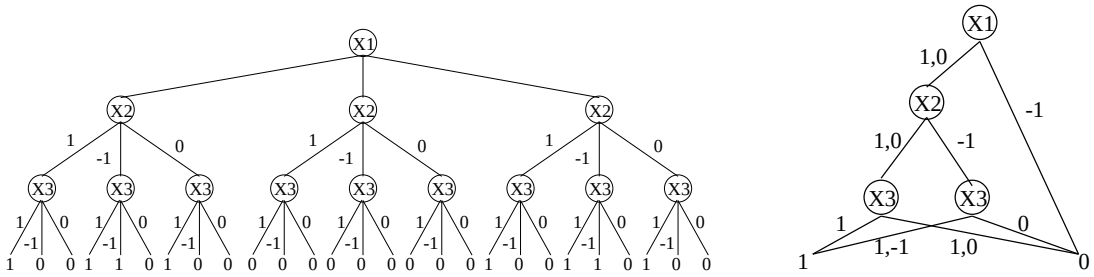
◇

We can then decompose all polynomial functions using the basis of monomials $e_1^{\alpha_1} \ldots e_n^{\alpha_n}$. Hence, given a polynomial function P in $A[X]$, and an order $X_1 \prec X_2 \prec \ldots \prec X_n$ on the variables, an h-expression of P is either $P(X) = c_1 e_i^1 + c_2 e_i^2 + c_3 e_i^3$ where $c_i \in \mathbb{Z}/_{3\mathbb{Z}}$, or $P(X) = e_i^1 P_1 + e_i^2 P_2 + e_i^3 P_3$ where the P_i are h-expressions with variables greater than X_i.

An h-expression may be pictured as a ternary tree. For example, the polynomial function

$$\begin{aligned} P(X_1, X_2, X_3) = {} & X_1^2 X_2^2 X_3^2 - X_1^2 X_2^2 X_3 - X_1^2 X_2 X_3^2 + X_1^2 X_2 X_3 \\ & - X_1^2 X_3^2 - X_1^2 X_3 - X_1 X_2^2 X_3^2 + X_1 X_2^2 X_3 + X_1 X_2 X_3^2 - X_1 X_2 X_3 \\ & + X_1 X_3^2 + X_1 X_3 + X_2^2 X_3^2 - X_2^2 X_3 - X_2 X_3^2 + X_2 X_3 - X_3^2 - X_3. \end{aligned}$$

is represented as in Figure 12(a), where the left child of a subtree with root X_i represents the e_i^1 factor, the middle child the e_i^2 factor and the right child the e_i^3 factor. The leaves are labeled with the numerical coefficients.



(a) An example of ternary tree (b) The reduced TDD

Figure 12: TDD representation of a polynomial function

Two ideas lead to an efficient implementation of polynomial functions: the first one is to reduce the h-expressions of the form $P(X) = e_i^1 P_1 \ + e_i^2 P_2 \ + e_i^3 P_3$ by eliminating the idempotents e_i^α when $P_1 = P_2 = P_3$ and by replacing the former expression by the common value P_1. The second idea comes from the fact that in the tree representation of h-expressions, many subtrees are identical. We obtain to a generalization of Bryant graphs [15] and fortunately these graphs retain in some way the orthogonality property with respect to polynomial operations. The graph representation of the previous example is given Figure 12(b).

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399